DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

# Structured Extraction of Terms and Conditions from German and English Online Shops

Tobias Michael Schamel

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

# Structured Extraction of Terms and Conditions from German and English Online Shops

# Strukturierte Extraktion von AGB aus deutschen und englischen Onlineshops

| | |
|---|---|
| Author: | Tobias Michael Schamel |
| Supervisor: | Prof. Dr. Florian Matthes |
| Advisor: | Dr. Daniel Braun |
| Submission Date: | 15.08.2021 |

I confirm that this bachelor's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.08.2021 Tobias Michael Schamel

# Acknowledgments

First of all, I would like to thank my advisor Dr. Daniel Braun, who has supported and advised me with full commitment over the last few months.
Thank you very much for your great personal commitment.

I would also like to thank Prof. Dr. Florian Matthes, who gave me the opportunity to write my bachelor's thesis at his chair *sebis*.

# Abstract

Terms and Conditions (T&C) of online shops are rarely read, even more rarely understood, and even still accepted by just about everyone - this behavior is also called *the biggest lie on the internet*, as only 0.1%-0.2% of users actually read T&Cs.

To encounter this behavior, which can be costly for a legal layman, an AI-supported evaluation of T&Cs can support the everyday user. The AI-supported evaluation of T&Cs from online shops requires a structured representation of the information that can usually be found on the shop's website.

In this thesis, we build a python library capable of extracting the main content of an online shop's T&C website, which includes removing navigation and footer bars. The main content will be segmented into paragraphs and structured as a tree representing the relationships of different paragraphs and clauses. We will identify and evaluate different approaches to fulfill these tasks.

A new content extraction algorithms called *Common Ancestor Extractor* is introduced in this thesis.

Keywords: Content Extraction, Hierarchy Extraction, Terms and Conditions, Common Ancestor Extractor, Online Shop, AGB-Check, Consumer Protection

Research Areas: Information Systems, Computer Science, NLP-Pipe, Content Extraction, Hierarchy Extraction

# List of Abbreviations

**T&C** Terms and Conditions
**SEO** Search Engine Optimization
**DOM** Document Object Model
**HTML** Hypertext Markup Language
**XPath** XML Path Language
**CSS** Cascading Style Sheet
**NLP** Natural Language Processing
**AI** Artificial Intelligence
**CMS** Content Management Systems
**MCS** Most Common Style
**URL** Uniform Resource Locator
**JSON** JavaScript Object Notation Extensible Markup Language
**XML** Extensible Markup Language

# Contents

**Acknowledgments** **iii**

**Abstract** **iv**

**List of Abbreviations** **v**

**1. Introduction** **1**
1.1. Context in AGB-Check . . . . . 1
1.2. Research Objectives . . . . . 2
1.3. Research Approach . . . . . 3

**2. Related Literature** **5**
2.1. Content Extraction . . . . . 5
2.2. Hierarchy Extraction . . . . . 8

**3. Requirements** **10**
3.1. Content Extraction . . . . . 10
3.1.1. Manual Review . . . . . 10
3.1.2. Evaluation of Existing Solutions . . . . . 11
3.1.3. Derived Requirements . . . . . 14
3.2. Hierarchy Extraction . . . . . 15
3.2.1. Manual Review . . . . . 15
3.2.2. Derived Requirements . . . . . 16

**4. Implementation** **18**
4.1. Architecture . . . . . 18
4.2. Additional Technology . . . . . 19
4.2.1. Web Page Download . . . . . 20
4.2.2. Hypertext Markup Language (HTML) Parser . . . . . 21
4.2.3. Sentence Segmentation and Tokenising . . . . . 22
4.2.4. Language Determination . . . . . 22
4.3. Document Object Model (DOM)-Tree . . . . . 23

4.4. Content Extraction . . . . . 24
4.4.1. Threshold . . . . . 26
4.4.2. Common Ancestor Extractor: Naive Style . . . . . 27
4.4.3. Common Ancestor Extractor: Rendered Style . . . . . 28
4.4.4. Common Ancestor Extractor: Naive Style and Short Text Exclusion 29
4.5. Hierarchy Extraction . . . . . 30
4.5.1. Content Blocks . . . . . 32
4.5.2. Visual Styles . . . . . 35
4.5.3. Enumerations . . . . . 35
4.6. Target Format . . . . . 39

**5. Evaluation** **40**
5.1. Content Extraction . . . . . 40
5.2. Hierarchy Extraction . . . . . 43

**6. Conclusion and Future Work** **48**

**A. Sample** **50**

**B. Processing Demo** **53**
B.1. Content Extraction . . . . . 55
B.2. Hierarchy Extraction . . . . . 58
B.3. Generating the Target Format . . . . . 62

**C. Library Documentation** **66**
C.1. Methods . . . . . 66
C.2. Related . . . . . 67

**List of Figures** **68**

**List of Tables** **69**

**Bibliography** **70**

# 1. Introduction

This chapter declares the context of this bachelor thesis in the context of the research project AGB-Check in Section 1.1. Following, the objectives and the corresponding research questions of the thesis are outlined in Section 1.2. Section 1.3 covers the methodology used to answer the various research questions outlined in Section 1.2.

## 1.1. Context in AGB-Check

Many people use the internet to shop online and almost all of them have accepted T&Cs that they have not read or understood before. This phenomenon is also called *"The biggest lie on the internet"* [17]. Bakos et al. discovered, that only 0.1% to 0.2% of online shoppers actually read the T&Cs before agreeing to legally binding purchase contracts [1].

The research project *AGB-Check* [6] (AI-Supported Legal Review of Terms and Conditions to Strengthen Consumer Protection) conducted by the sebis chair is currently evaluating the possibility of making legal expertise available more quickly and cheaply. This would give consumers, who are mostly legal laypersons, the possibility to make informed decisions and thereby strengthen consumer protection.

The research team of AGB-Check is currently able to determine the Uniform Resource Locator (URL) of an online shop's T&Cs and process the structured data extracted from the T&Cs. They are still lacking a reliable solution for content extraction from websites in the domain of T&Cs. The broad range of existing content extractors is built for newspaper articles and other more common content sources. They solely extract the content without retaining the hierarchical structure of the text.

The German civil code[1] (§305 - §310) requires T&Cs to contain certain clauses, as well as a clear logical structure to ease understanding. This fortifies the importance of extracting the hierarchical structure in a legal context, as the structure effects the legal validity of T&C. Furthermore, checking for required clauses during the Artificial Intelligence (AI)-supported evaluation is much easier when the structure of the contract

[1] https://www.gesetze-im-internet.de/englisch_bgb/index.html

is kept.

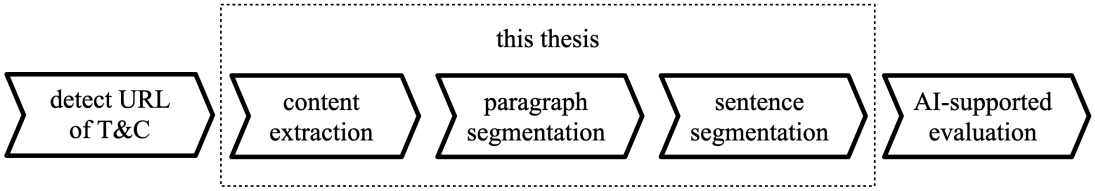



Figure 1.1.: The Context of this thesis in the project AGB-Check.

## 1.2. Research Objectives

The previously described gap (see Figure 1.1) in the Natural Language Processing (NLP) pipe must be filled by a reliable solution. Possible errors will otherwise propagate through the entire pipe making the result useless. Before a solution can be reliably developed, the prevailing problems of existent solutions need to be understood and the specific characteristics of T&Cs need to be investigated.

The goal of this thesis is to develop the necessary reliant solution to fill the gap in the pipe. The developed library takes the URL, extracts content and hierarchy, and converts it into a structured representation previously defined by the team of AGB-Check.

The research questions and the rationale underlying them are explained below.

**RQ1:** What are special requirements for the extraction of contracts in comparison with *regular* (i.e. news articles, blogposts) content?

Existing solutions for content extraction focus on regular content like newspaper articles most of the time. The differences between newspaper articles and contracts have to be identified to understand where existing content extractors fail in the case of T&Cs.

**RQ2:** How to extract the relevant parts from contracts in raw HTML?

The knowledge derived from *RQ1* can be used to develop a content extractor that is tailored to extract T&Cs and achieves significantly more accurate results in its domain compared to generic content extractors.

**RQ3:** How to extract the structure and the hierarchy of paragraphs, (sub-)titles, and related clauses?

Contractual documents are usually required to be divided into paragraphs which themselves are structured in a hierarchical way resulting in a tree structure. This tree structure has to be preserved as it allows to draw conclusions based on the relation of multiple clauses during the AI-supported evaluation. In addition, the individual paragraphs can be preceded by a headline. These headings must also be detected and retained for later processing.

As the input to a content extractor is a website, it must be studied, which existing HTML and Cascading Style Sheet (CSS) properties provide useful information on hierarchical properties and can thus be used to structure the content.

## 1.3. Research Approach

This section describes the research approach used to answer the research questions depicted in Section 1.2.

Figure 1.2 shows the flow of knowledge between the research questions.

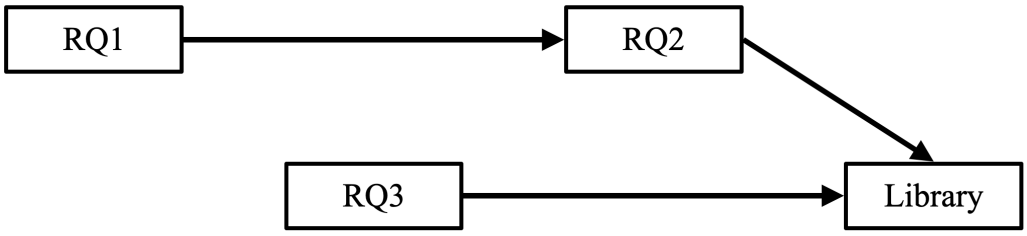



Figure 1.2.: The relationship and flow of knowledge between the three research questions.

**RQ1: What are special requirements for the extraction of contracts in comparison with *regular* (i.e. news articles, blogposts) content?**

The first research questions will be answered by evaluating the results of existing content extractors. This allows for an analysis of the current problems that will be

addressed by the solution developed in this thesis. Clusters of common mistakes need to be identified to deduce special requirements for the extraction of contracts.

**RQ2: How to extract the relevant parts from contracts in raw HTML?**

The knowledge gained from RQ1 will be used to develop a reliable python library that can be used to extract the relevant parts from contracts in raw HTML.

**RQ3: How to extract the structure and the hierarchy of paragraphs, (sub-)titles, and related clauses?**

First, the implementation of hierarchical structures in T&Cs must be analyzed. A sample of web pages will be reviewed manually to retrieve information on the HTML structure, the actual numeration (if existing), and the CSS styling.

Based on this knowledge, the hierarchical relationship of different paragraphs can be extracted.

**StructuredLegalExtraction Library**

Results from RQ2 and RQ3 will be used to implement a python library for the extraction of T&Cs from online shops. This library will be evaluated and compared to existing solutions.

# 2. Related Literature

In this chapter, the related literature in the area of Content Extraction (i.e. the identification and extraction of the main content) and Hierarchy Extraction (i.e. the identification and preservation of the T&Cs hierarchy) is summarized to get an overview of existing approaches to accomplish these tasks.

## 2.1. Content Extraction

Extracting content in the domain of Terms and Conditions (T&Cs) from websites has not been studied so far. Previous research has rarely looked at the problem in a domain-specific way. There is some research in the domain of news articles and more research in generic content extraction, as most solutions were evaluated using the dataset of the *Cleaneval* competition [3]. Existing solutions were mainly used in the context of natural language processing [12, 10, 9, 14] (i.e. corpus construction). Additionally, researchers tried to find ways to re-display relevant information on small screens (i.e. smartphones) [10, 19, 7]. Rationale of generic content extraction, as well as some components of customized scripts used to extract content from a particular website can be adopted to improve the outcome of this thesis.

According to Kilgariff, the task of creating web corpora involves "crawling, downloading, 'cleaning' and de-duplicating the data" [12]. The first three tasks will be crucial for the extraction of T&Cs of online shops. The most essential operation "consists in retaining the desired content while discarding the rest" [2] ("cleaning"). Considering the early point cleaning takes in a pipe, it becomes clear how errors that occur here propagate through all following steps [19]. There is a number of synonyms to the process of "cleaning" like *boilerplate detection/removal* [13], *template matching* [21] and *content identification/extraction* [10, 2].

Gibson et al. [10] described the process of cleaning as a Sequence Labeling problem on a document broken down into a sequence of blocks. Each block needs to be classified as either *Content* or *NotContent*. Kohlschütter et al. [13] work with a more detailed four-class separation. Another more risky approach is a Boundary Detection Method where a heuristic needs to determine a *Start*- and *End*-Block framing the

whole content, i.e. everything between *Start-* and *End*-Block is considered *Content* whereas everything else is discarded as *NotContent*. Gibson et al. also stated that the Boundary Detection Method did not perform any better than the worst Sequence Labeling approach. This might be explained by their dataset created from news articles. The focus on news articles means that a high proportion of the text is interspersed with images, advertisements, or recommendations for further content which can weaken the performance.

According to Viveros-Jiménez et al., the classification needs to take both HTML structure and the actual content into account. "Purely text-based or purely HTML-based approaches do not have perfect results." [11]

Several approaches for classifying blocks as *Content* or *NotContent* can be found in the literature.

Kohlschütter et al. [13] propose a classification inspecting the text on a functional level using a set of so-called shallow text features. Shallow text features are statistical calculations on block-level looking at domain and language independent features like link density ($\frac{\text{amount of letters in <a> tags}}{\text{total amount of letters}}$), the average sentence length, the uppercase ratio, etc. In Addition, densiometric features like the text density are also taken into account. These features can be used in a trained classification model. This approach is implemented in the popular Boilerpipe[1] algorithm. Kohlschütter et al. interpreted some of their findings from a linguistic perspective. They state that full sentences supported by grammatical constructs often belong to the descriptive *Content* of a website. Short text sequences made of grammatically incomplete and simple sentences (e.g., "Contact us", "Read more") are understood by the audience without large effort. Thus it is often found in headlines or navigational text classified as *NotContent*.

JusText [20] uses a low amount of features to determine the likelihood of a block being *Content* or *NotContent*. Uncertainties are dealt with within the next step, which involves an analysis of the relative position of a block in the HTML tree including the classification of its neighbors. This step is based on the assumption that *Content* blocks are to be found near other *Content* blocks (and vice versa). This approach is implemented in the JusText[2] library. Usage of the JusText library requires the determination of the language, as different types of words need to be classified to determine the likelihood of a paragraph being a headline.

Viveros et al. [11] introduce an improvement to the Boilerpipe algorithm by also taking the HTML tree structure into account. They use the Boilerpipe algorithm to select a set

[1] https://code.google.com/p/boilerpipe
[2] https://code.google.com/archive/p/justext/

of mostly relevant content which can then be filtered for other non-informative content to be removed using the HTML tree assuming that *Content*- and *NoContent*-blocks create clusters. The filtering procedure works as follows:

1. Identify and select all paragraphs (paragraphs are defined by a number of tags not containing paragraphs themselves).
2. Generate a list of ancestors for all selected paragraphs.
3. Group all paragraphs sharing the same $n$-th paragraph (experiments showed that $n = 2$ generated the best results).
4. Select the group containing the highest amount of text and discard everything else.

Pasternack and Roth [19] try to solve the task by finding a maximum subsequence in tokenized HTML documents. Each token (including HTML tags) is assigned a positive or negative score determined by token-level classifiers. Pasternack and Roth evaluated different classifiers, e.g. simply assigning predefined scores to words and tags. They also investigated more advanced classifiers which combined Naive Bayes classification with features of the surrounding tags.

Debnath et al. [8] propose a content extraction algorithm which is looking at similar web pages from a class. A class is defined as a group of pages derived from the same script or the same website. Recurring and similar elements are likely to be non-informative and thus classified as *NoContent*.

Cai et al. [7] implemented a content extraction method based on visual representation, as web pages are usually divided into different regions by visual features to make them readable. Approaches using the visual representation are robust in environments, where HTML structure and layout structure differ a lot. Cai et al. propose a top-down algorithm to separate the website, respectively its vision-based content structure, into layout blocks ($O$). Layout blocks on the same depth of the derived layout tree a collected in a pool. Within these pools, another algorithm detects all the visual separators (horizontal or vertical lines that do not cross any of the blocks visually, $\Phi$) and assigns a weight ($\delta$) based on the difference between neighboring blocks. This results in a triple representation of the web page $\Omega = \{O, \Phi, \delta\}$, where $O$ is a set of sub web pages.

Sano et al. [21] defined eight layout templates including information about the position of the main content. They identify the best-matching layout template by determining a layout structure using a method similar to the one proposed by Cai et al. Once a layout template is selected, the affiliated main content, respectively its block, is extracted.

Evert [9] analyzed the use of simple character-level n-gram models to distinguish

between *Content* and *NoContent*. Therefore, no information about the HTML tree or tags were used. After the whole plain text is extracted from a web page, two distinct character-level n-gram language models are used for *Content* and *NoContent*. The approach is implemented in *NCleaner*[3].

## 2.2. Hierarchy Extraction

The content extracted for *AGB-Check* needs to be hierarchically structured (RQ3). Research on extraction of hierarchical structures is scarce.

Manabe and Tajima [16] introduced a segmenting method extracting the hierarchical structure of HTML documents based on the differences of the visual styles in hierarchical headings. They defined four properties to identify headings, their corresponding blocks, and their hierarchical relationships based on the way humans read hierarchically structured content:

1. headings appear at the beginning of the corresponding blocks
2. headings are given prominent visual styles
3. headings of the same level share the same visual style
4. headings of higher levels are given more prominent visual styles

According to them, the nested hierarchy of an HTML document can contribute some information but does not necessarily coincide with the actual hierarchical structure of an HTML document. The HTML tags `<h1>` to `<h6>` - originally meant to structure a document and indicate headings - are often misused for Search Engine Optimization (SEO) or not used at all. The reference implementation of this algorithm is publicly available[4].

Okada and Arakawa [18] used the J48 algorithm to derive extraction rules (decision tree) based on 26 HTML attributes.

Sano et al. [21] used a similar approach with only nine parameters. The parameters are chosen with respect to the following characteristics of headings:

1. headings have few child nodes
2. headings have a short text length
3. the width of headings is greater than their height
4. the size of a heading is smaller than the size of the following content block

[3] https://sourceforge.net/projects/webascorpus/files/NCleaner/NCleaner-1.0/
[4] https://github.com/tmanabe/HEPS

underneath it

# 3. Requirements

This chapter aims at identifying special requirements for T&C extraction in comparison with *regular* content. In the domain of content extraction, this was done by analyzing the structure of webpages by manual review. Besides this, the results of existing solutions were compared with the expected output gathered by hand. This allows the derivation of common sources of errors in the extraction process and by this requirements for the extraction of T&Cs. Requirements for hierarchy extraction were solely identified by a manual review.

For evaluation purposes, a sample of 30 German and 20 English T&C pages from the data set by Braun and Matthes [5] which is available under CC BY-SA 3.0 license from GitHub[1] is used. The quality of the data from the English sample was partly less good. 15% of the sample did not refer to the correct T&C page of the respective online shop. These links were corrected manually.

## 3.1. Content Extraction

### 3.1.1. Manual Review

In a manual review process similarities among T&C pages were identified. While news articles are often interrupted by advertisements or references to similar articles, in all of the cases examined in the sample, T&Cs are uninterrupted text on a rather simply structured page. This makes the domain-specific content extraction of T&Cs less challenging than generic content extractions presented in Section 2.1, as there is no need to reclassify content between the presumed start and the presumed end of the presumed content.

However, the content is not always grouped within a single large paragraph. In the German sample, bisection of the content can often be observed. The T&Cs are divided into *1. Allgemeine Geschäftsbedingungen* and *2. Kundeninformation* where both contain relevant information. As described later, some extractors had problems with this,

[1] https://github.com/sebischair/TC-Detection-Corpus/

as they only extracted one of the paragraphs. The German T&Cs are by far more structured than the English ones due to tighter regulation. This might explain why many German online shops used a template by Haendlerbund.

Usually, the relevant content shared a common style and it could be found in the same depth of an HTML document. In rare cases, this style was also used in the footer or the header of an HTML document. Some exceptions to these observations can be found on those pages including a different styled withdrawal form. As shown in Figure 3.1, in some cases the withdrawal form or the withdrawal information are highlighted by a border and a different background color.

The form is often composed of underscores, blank spaces, and text. In some cases, it can be downloaded as a PDF using a link attached to the T&Cs.

### 3.1.2. Evaluation of Existing Solutions

To evaluate the existing solutions, tests were carried out with each of the individual web pages from the sample. The texts extracted by the respective Python libraries were compared with the manually extracted texts using a DIFF checker[3].

In order to evaluate the results of the test, the following aspects were examined:

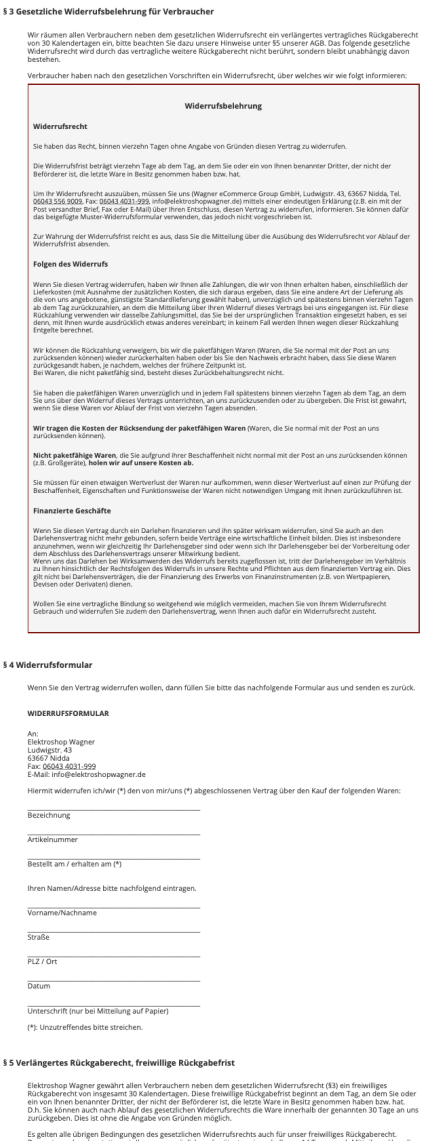
- start: *correct*, *too early*, *too late*
- end: *correct*, *too early*, *too late*
- center: *correct*, *short* (links & addresses), *long* (whole paragraphs or sentences), *short & long*
- language: *de*, *en*
- additional remarks: used to identify further patterns in the results

Requirements for the *correct* identification of the content were relaxed, as the importance of extracting a meaningless headline on top of the page is low.

Tables 3.1, 3.2, and 3.3 summarize the results of the examination. The worst (i.e. the one losing the most content) property of the indicators used is colored red, the best property is colored green. 6 of 20 sampled web pages from the English sample were not extractable due to malformed HTML. As small mistakes in the HTML structure are not uncommon and are usually fixed by browsers rendering them, this should not be the case.

[2]https://www.elektroshopwagner.de/shop_content.php/coID/3/content/Unsere-AGB/ last accessed on 20210613

[3]https://www.diffchecker.com/diff/



§ 3 Gesetzliche Widerrufsbelehrung für Verbraucher

Wir räumen allen Verbrauchern neben dem gesetzlichen Widerrufsrecht ein verlängertes vertragliches Rückgaberecht von 30 Kalendertagen ein, bitte beachten Sie dazu unsere Hinweise unter §5 unserer AGB. Das folgende gesetzliche Widerrufsrecht wird durch das vertragliche weitere Rückgaberecht nicht berührt, sondern bleibt unabhängig davon bestehen.

Verbraucher haben nach den gesetzlichen Vorschriften ein Widerrufsrecht, über welches wir wie folgt informieren:

**Widerrufsbelehrung**

**Widerrufsrecht**

Sie haben das Recht, binnen vierzehn Tagen ohne Angabe von Gründen diesen Vertrag zu widerrufen.

Die Widerrufsfrist beträgt vierzehn Tage ab dem Tag, an dem Sie oder ein von Ihnen benannter Dritter, der nicht der Beförderer ist, die letzte Ware in Besitz genommen haben bzw. hat.

Um Ihr Widerrufsrecht auszuüben, müssen Sie uns (Wagner eCommerce Group GmbH, Ludwigstr. 43, 63667 Nidda, Tel. 06043 556 9003, Fax: 06043 4031-999, info@elektroshopwagner.de) mittels einer eindeutigen Erklärung (z.B. ein mit der Post versandter Brief, Fax oder E-Mail) über Ihren Entschluss, diesen Vertrag zu widerrufen, informieren. Sie können dafür das beigefügte Muster-Widerrufsformular verwenden, das jedoch nicht vorgeschrieben ist.

Zur Wahrung der Widerrufsfrist reicht es aus, dass Sie die Mitteilung über die Ausübung des Widerrufsrecht vor Ablauf der Widerrufsfrist absenden.

**Folgen des Widerrufs**

Wenn Sie diesen Vertrag widerrufen, haben wir Ihnen alle Zahlungen, die wir von Ihnen erhalten haben, einschließlich der Lieferkosten (mit Ausnahme der zusätzlichen Kosten, die sich daraus ergeben, dass Sie eine andere Art der Lieferung als die von uns angebotene, günstigste Standardlieferung gewählt haben), unverzüglich und spätestens binnen vierzehn Tagen ab dem Tag zurückzuzahlen, an dem die Mitteilung über Ihren Widerruf dieses Vertrags bei uns eingegangen ist. Für diese Rückzahlung verwenden wir dasselbe Zahlungsmittel, das Sie bei der ursprünglichen Transaktion eingesetzt haben, es sei denn, mit Ihnen wurde ausdrücklich etwas anderes vereinbart; in keinem Fall werden Ihnen wegen dieser Rückzahlung Entgelte berechnet.

Wir können die Rückzahlung verweigern, bis wir die paketfähigen Waren (Waren, die Sie normal mit der Post an uns zurücksenden können) wieder zurückerhalten haben oder bis Sie den Nachweis erbracht haben, dass Sie diese Waren zurückgesandt haben, je nachdem, welches der frühere Zeitpunkt ist.
Bei Waren, die nicht paketfähig sind, besteht dieses Zurückbehaltungsrecht nicht.

Sie haben die paketfähigen Waren unverzüglich und in jedem Fall spätestens binnen vierzehn Tagen ab dem Tag, an dem Sie uns über den Widerruf dieses Vertrags unterrichten, an uns zurückzusenden oder zu übergeben. Die Frist ist gewahrt, wenn Sie diese Waren vor Ablauf der Frist von vierzehn Tagen absenden.

**Wir tragen die Kosten der Rücksendung der paketfähigen Waren** (Waren, die Sie normal mit der Post an uns zurücksenden können).

**Nicht paketfähige Waren**, die Sie aufgrund ihrer Beschaffenheit nicht normal mit der Post an uns zurücksenden können (z.B. Großgeräte), **holen wir auf unsere Kosten ab.**

Sie müssen für einen etwaigen Wertverlust der Waren nur aufkommen, wenn dieser Wertverlust auf einen zur Prüfung der Beschaffenheit, Eigenschaften und Funktionsweise der Waren nicht notwendigen Umgang mit ihnen zurückzuführen ist.

**Finanzierte Geschäfte**

Wenn Sie diesen Vertrag durch ein Darlehen finanzieren und ihn später wirksam widerrufen, sind Sie auch an den Darlehensvertrag nicht mehr gebunden, sofern beide Verträge eine wirtschaftliche Einheit bilden. Dies ist insbesondere anzunehmen, wenn wir gleichzeitig ihr Darlehensgeber sind oder wenn sich ihr Darlehensgeber bei der Vorbereitung oder dem Abschluss des Darlehensvertrags unserer Mitwirkung bedient.
Wenn uns das Darlehen bei Wirksamwerden des Widerrufs bereits zugeflossen ist, tritt der Darlehensgeber im Verhältnis zu ihnen hinsichtlich der Rechtsfolgen des Widerrufs in unsere Rechte und Pflichten aus dem finanzierten Vertrag ein. Dies gilt nicht bei Darlehensverträgen, die der Finanzierung des Erwerbs von Finanzinstrumenten (z.B. von Wertpapieren, Devisen oder Derivaten) dienen.

Wollen Sie eine vertragliche Bindung so weitgehend wie möglich vermeiden, machen Sie von Ihrem Widerrufsrecht Gebrauch und widerrufen Sie zudem den Darlehensvertrag, wenn Ihnen auch dafür ein Widerrufsrecht zusteht.

§ 4 Widerrufsformular

Wenn Sie den Vertrag widerrufen wollen, dann füllen Sie bitte das nachfolgende Formular aus und senden es zurück.

**WIDERRUFSFORMULAR**

An:
Elektroshop Wagner
Ludwigstr. 43
63667 Nidda
Fax: 06043 4031-999
E-Mail: info@elektroshopwagner.de

Hiermit widerrufen ich/wir (*) den von mir/uns (*) abgeschlossenen Vertrag über den Kauf der folgenden Waren:

Bezeichnung

Artikelnummer

Bestellt am / erhalten am (*)

Ihren Namen/Adresse bitte nachfolgend eintragen.

Vorname/Nachname

Straße

PLZ / Ort

Datum

Unterschrift (nur bei Mitteilung auf Papier)

(*): Unzutreffendes bitte streichen.

§ 5 Verlängertes Rückgaberecht, freiwillige Rückgabefrist

Elektroshop Wagner gewährt allen Verbrauchern neben dem gesetzlichen Widerrufsrecht (§3) ein freiwilliges Rückgaberecht von insgesamt 30 Kalendertagen. Diese freiwillige Rückgabefrist beginnt an dem Tag, an dem Sie oder ein von Ihnen benannter Dritter, der nicht der Beförderer ist, die letzte Ware in Besitz genommen haben bzw. hat. D.h. Sie können auch nach Ablauf des gesetzlichen Widerrufsrechts die Ware innerhalb der genannten 30 Tage an uns zurückgeben. Dies ist ohne die Angabe von Gründen möglich.

Es gelten alle übrigen Bedingungen des gesetzlichen Widerrufsrechts auch für unser freiwilliges Rückgaberecht.

Figure 3.1.: Withdrawal information and withdrawal form highlighted by border and different background color. Extracted from Elektroshopwagner[2].

### Boilerpipe

The three examined Boilerpipe extractors (*ArticleExtractor*, *LargestContentExtractor*, *CanolaExtractor*) showed significantly different results. Problems with links and addresses (*short*) occurred frequently with the *ArticleExtractor* and the *CanolaExtractor*.

| | too late | too early | correct |
|---|---|---|---|
| Boilerpipe ArticleExtractor | 16 | 4 | 24 |
| Boilerpipe LargestContentExtractor | 29 | 2 | 13 |
| Boilerpipe CanolaExtractor | 7 | 15 | 22 |
| JusText | 6 | 11 | 25 |
| Trafilatura | 5 | 2 | 37 |

Table 3.1.: Performance of detecting the start of the T&Cs for each examined extractor.

| | too early | too late | correct |
|---|---|---|---|
| Boilerpipe ArticleExtractor | 23 | 13 | 8 |
| Boilerpipe LargestContentExtractor | 32 | 1 | 11 |
| Boilerpipe CanolaExtractor | 3 | 27 | 14 |
| JusText | 8 | 15 | 19 |
| Trafilatura | 5 | 4 | 35 |

Table 3.2.: Performance of detecting the end of the T&Cs for each examined Extractor.

With *ArticleExtractor*, it often happened that the end of the content was recognized way too early (see Table 3.2). This was often due to addresses, enumerations, or withdrawal forms in the content. Another reason for this is that German T&Cs are often divided into two parts, where only the first one was detected and extracted by the *ArticleExtractor*. Similar behavior was observed for the beginning of the content, where address information etc. were omitted regularly. Since the *ArticleExtractor* often stopped/started the extraction here, the results for the middle are better than with other extractors (see Table 3.3).

The *CanolaExtractor* was trained on the KrdWrd Canola corpus[4]. It recurrently extracted cookie information or web page footer, i.e. extracted too much content. The main content was usually detected but the extracted text was interrupted many times, as can be seen in Table 3.3. This applies to some short paragraphs, headings, and as with the *ArticleExtractor*, also addresses and withdrawal forms.

The *LargestContentExtractor* extracts a continuous piece of content in all cases. This connected part is not a single HTML node but a consecutive series of HTML nodes. Within this continuous excerpt, the clauses are not classified any further, which is why there are no interruptions (see Table 3.3).

[4]https://krdwrd.org/

| | short | long | short & long | correct |
|---|---|---|---|---|
| Boilerpipe ArticleExtractor | 3 | 4 | 3 | 33 |
| Boilerpipe LargestContentExtractor | 0 | 0 | 0 | 44 |
| Boilerpipe CanolaExtractor | 1 | 8 | 33 | 0 |
| JusText | 11 | 7 | 21 | 5 |
| Trafilatura | 1 | 5 | 0 | 38 |

Table 3.3.: Performance of detecting the center of the T&Cs for each examined Extractor.

#### JusText

The results of *JusText* were similar to the results of the *Boilerpipe CanolaExtractor*. The main content was usually detected but the extracted text was interrupted (see Table 3.3), as *JusText* classified many headlines, addresses, and paragraphs containing links as boilerplate.

#### Trafilatura

*Trafilatura* produced the best results of the examined solutions. However, *Trafilatura* seemed to have problems with the headlines of paragraphs, as it ignored some of them. In addition, there were problems with content that was not directly visible to a user but had to be unfolded in the browser manually.

### 3.1.3. Derived Requirements

The following requirements for the task of content extraction can be derived from the manual review and the evaluation of existing solutions:

1. extract (largest) continuous part of HTML document
2. extract the content sharing a common style and depth in HTML tree
3. extract the withdrawal form/information with different style and different depth which are part of the relevant content
4. extract address information
5. always extract both of the sections *1. Allgemeine Geschäftsbedingungen* and *2. Kundeninformation*
6. extract *invisible* content which needs to be unfolded in the browser

## 3.2. Hierarchy Extraction

### 3.2.1. Manual Review

Data on the visual and HTML-based representations of the hierarchical structure of web pages containing T&Cs were gathered in a manual review. The results gained from the English sample and the German sample differed slightly. The following hierarchy representation classes were defined:

- Graphical: page is structured using graphical separators and different text styles
- Numeric: page is structured using a numeric scheme (Arabic, Latin/Roman, alphabetic, etc.)
- Numeric & Graphical: a combination of both *Graphical* and *Numeric* elements are used to structure the page

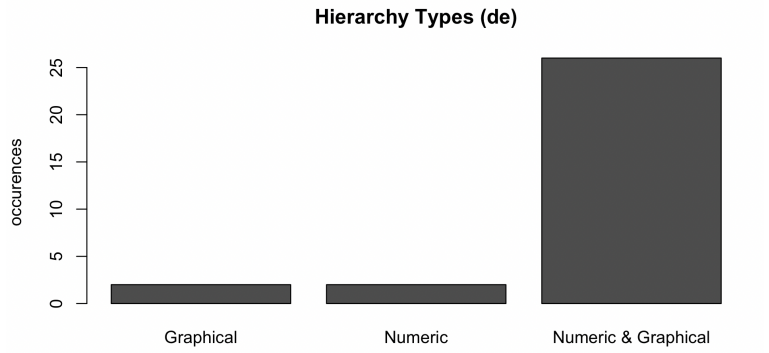



Figure 3.2.: Occurrences of different hierarchy styles in the German sample.

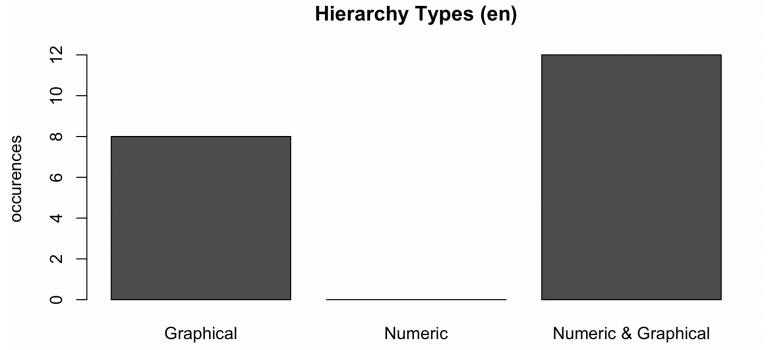



Figure 3.3.: Occurrences of different hierarchy styles in the English sample.

As shown in Figure 3.2, German online shops were much more likely to have structured their T&Cs with a combination of graphic and numeric elements. About 40% (see Figure 3.3 of online shops from the English sample only use graphical elements to represent the hierarchy of their T&Cs. The combination of numerical and graphical structuring in English online shops is mostly used if a clause is followed by multiple subclauses. In some cases each of the subclauses was grouped into its own HTML tag, in other cases, the subclauses were simply separated by linebreaks (`<br>`).

Most of the times enumeration patterns occurred, the enumeration was combined with an additional headline to the paragraph it precedes.

Some T&C pages contain a table of content similar to the one presented in Figure 3.4 with enumerations patterns like in the actual content, where these patterns can provide useful information on hierarchy and potential headlines. Lines containing enumerations can often be classified as headlines. However, in the leaves of the hierarchy tree, enumerations precede the actual content without an additional headline in some cases.

Enumerations occurring in a table of contents are not relevant for the actual hierarchy extraction and must thus be ignored in the process of hierarchy extraction to produce meaningful results.

### 3.2.2. Derived Requirements

The following requirements for the task of hierarchy extraction can be derived from the manual review:

1. extract subclauses grouped in their own paragraph forming HTML tag
2. detect numeration patterns (alphabetic, Arabic, Latin/Roman, section sign, etc.)
3. extract styling (CSS) of titles to determine associated content
4. ignore enumerations for the table of contents

[5]https://www.wexphotovideo.com/help/terms-and-conditions/ last accessed on 20210718

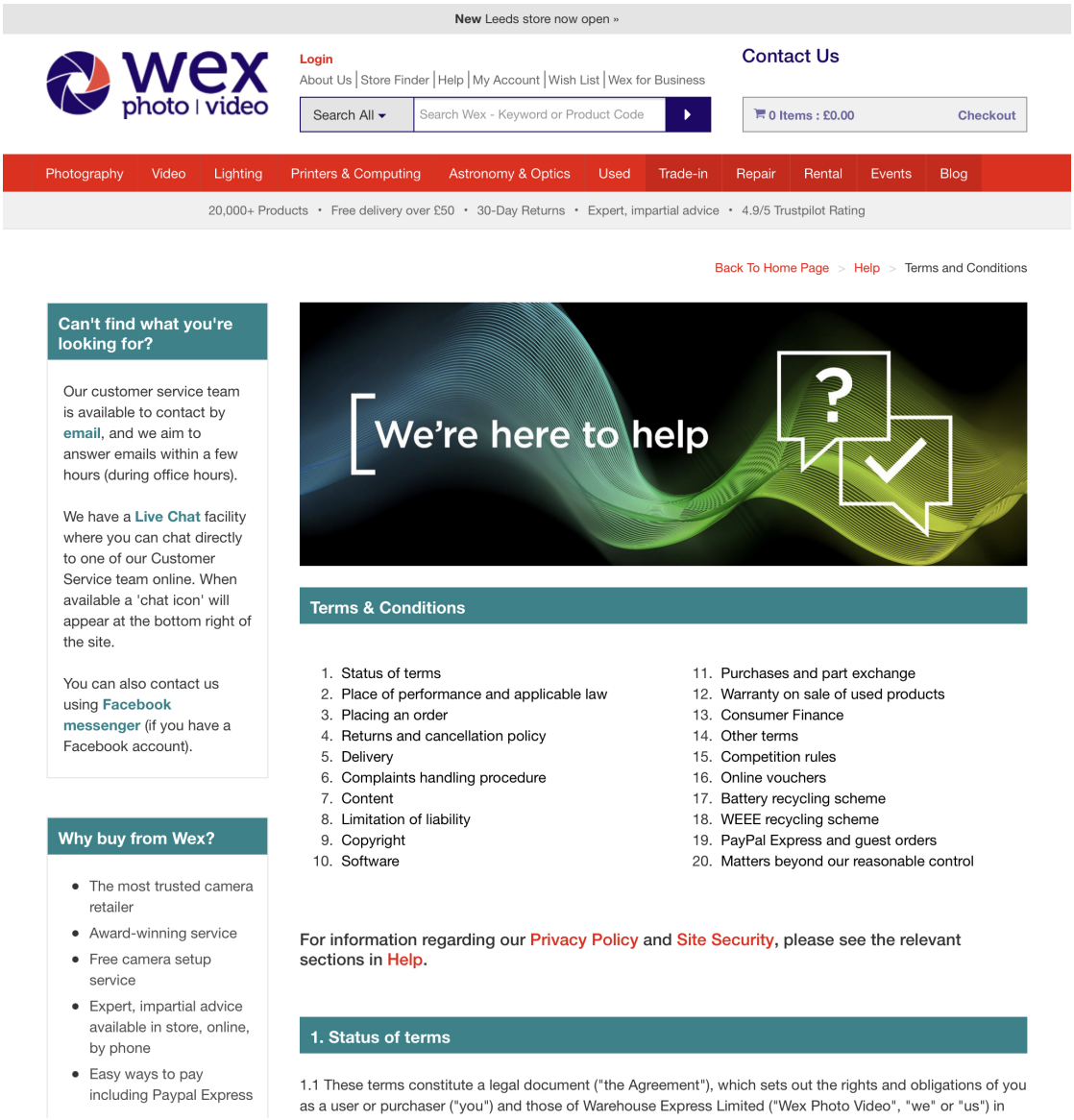



Figure 3.4.: Table of content with enumerations as a preface of T&Cs at Wex Photo Video[5].

# 4. Implementation

This chapter covers the software architecture, python libraries used in the implementation, and the actual implementation of the library used for content and hierarchy extraction. Characteristics specific to websites containing T&Cs were described in Chapter 3 and are used during this chapter to implement a domain-specific content extraction for T&Cs.

## 4.1. Architecture

The library consists of 2 main components, which serve to extract the content and structure it hierarchically. In addition, there is an auxiliary component that serves to download the web page and one that transforms the HTML content into a DOM-tree which is by far more useful for further processing. Another auxiliary component is used to generate the target structure relying on another auxiliary component to segment sentences.

The components are loosely coupled exposing only one main function each (except for the *Downloader*). This results in some kind of pipe architecture, where the processing components rely on a small number of auxiliary services. The whole pipe is covered by a façade which itself is exposing the main functionality (extracting content and hierarchy of T&C).

At first, the content is downloaded using the *Downloader* component. The downloaded content is processed by the *DOMParser* to achieve the desired DOM-tree holding no more information than those actually needed in this project.

The *ContentExtractor* component is responsible for detecting and extracting the main content of the downloaded page based on the parsed DOM-tree.

The *HierarchyExtractor* component uses the main content node of the DOM-tree to build its hierarchy tree based on visual information (CSS attached to DOM-node) and numerical patterns in the text.

In the last step, the hierarchy tree needs to be transformed to the target format by the *TargetFormat* component, which uses the *SentenceSegmenter* component to tokenize and

segment the content.

The architecture is also presented in Figure 4.1.

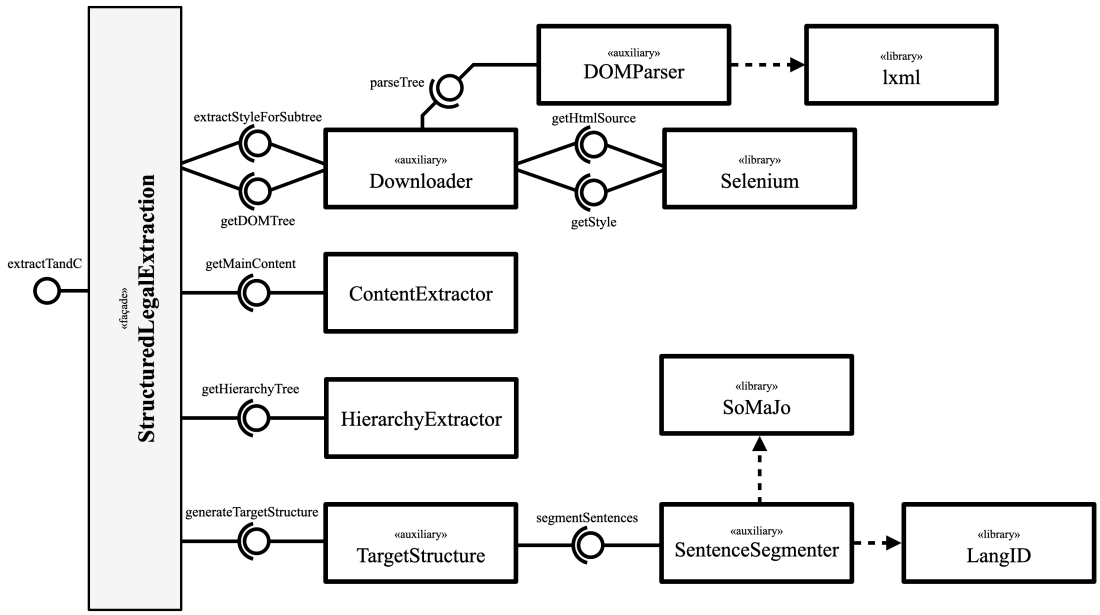



Figure 4.1.: Architecture of the StructuredLegalExtraction library.

The flow of data through the processing pipe is shown in the sequence diagram in Figure 4.2.

## 4.2. Additional Technology

Some publicly available libraries are used to pursue more generic tasks. This includes, among others, downloading the website with *Selenium* and tokenizing the sentences using *SoMaJo*.

In the following sections, the requirements (including *opt*imal features) to the utilized libraries are presented. This is followed by a description of the selected library and the alternatives investigated in the process of this thesis.

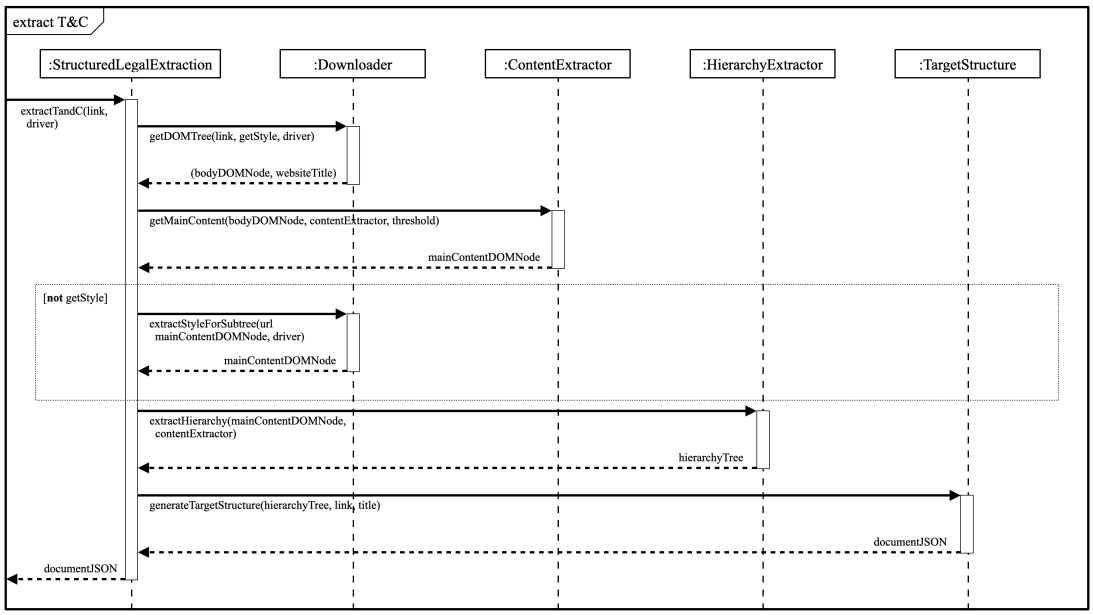



Figure 4.2.: Sequence diagram modelling the flow of information through the StructuredLegalExtraction library.

### 4.2.1. Web Page Download

In an effort to determine the main content of a web page, the entire content must first be downloaded. The following requirements have been identified:

- download full HTML file representing the web page
- extract CSS style information on the content
- full XML Path Language (XPath) support to navigate the *dom*-tree
- Java-Script support (*opt*)

#### Selenium

*Selenium*[1] is a well-known library usually utilized for website testing. The library controls a regular browser (multiple drivers are available) and is, therefore, able to interact with the website and download content. Selenium supports XPath to navigate the DOM-tree. However, the Selenium method `.find_by_xpath (path)` does not

[1] https://github.com/SeleniumHQ/selenium/

support text elements. Text can be extracted by accessing an element's `.text` attribute. Unfortunately, text segments (complete text - direct and indirect children - under the current node) cannot be mapped to the individual nodes so that one could track down which text has which style. This will require another library.

One of the outstanding features of Selenium compared to other libraries capable of downloading websites is the possibility to render CSS style information for nodes in the DOM-tree. This is a major requirement to process different styles, which helps to determine the hierarchy of paragraphs and to identify headlines. Selenium also supports Java-Script by rendering it in the browser controlled by the library.

#### Alternatives

Another investigated library, *requests-html*[2], supports Java-Script and XPath but could not extract styles. The use of the standard python *requests*[3] library by the *Python Software Foundation* does not support Java-Script, nor CSS style rendering. The extracted HTML needs to be processed using an additional HTML-parser.

### 4.2.2. HTML Parser

As Selenium is used as a web page downloader while offering no XPath support for text nodes, there is a need for a dedicated parser with the following requirements:

- full XPath support to navigate the DOM-tree
- XPath generation from elements

#### lxml

*lxml*[4] is an Extensible Markup Language (XML)-parser offering a special HTML-parser. Due to its full XPath support, it also allows extracting all child nodes including the text nodes of a DOM-node using the XPath `child::node()`. The library also allows generating XPath for elements relative to a given parent element which makes it a perfect fit to link with other libraries capable of XPath for style extraction.

[2]https://github.com/psf/requests-html/
[3]https://github.com/psf/requests/
[4]https://github.com/lxml/lxml/

#### Alternatives

Another well-known library used for XML (i.e. HTML) parsing is *BeautifulSoup*[5]. BeautifulSoup does not support tree-navigation using XPath which makes it more difficult to combine with other libraries (e.g. Selenium for style information). Navigation in BeautifulSoup is mainly accomplished using tag names or a DOM-node's relatives' attributes (`.parent`, `.next_sibling`, `.descendants`, etc.).

### 4.2.3. Sentence Segmentation and Tokenising

The content of the document needs to be segmented into sentences which themselves need to be tokenized into words for the target format. This requires reliable segmenting and tokenizing with respect to text elements like references to laws or address information common in the domain of T&Cs.

#### SoMaJo

*SoMaJo*[6] is used to segment sentences and tokenize them according to the desired target structure using a rule-based approach. SoMaJo needs information about the language of the content so that language-specific features can be taken into account when segmenting and tokenizing. According to Braun, SoMaJo performed best in the domain of T&Cs [4].

#### Alternatives

The *Natural Language Tool Kit (NLTK)*[7] and *spaCy*[8] performed worse in sentence segmentation, according to Braun [4]. They had problems with the domain-specific enumeration of paragraphs, address information, and laws, which often resulted in a sentence being divided into multiple sentences.

### 4.2.4. Language Determination

As the chosen library for sentence segmentation and tokenizing requires knowledge of the language of the text to account for language-specific abbreviations, etc., there is a

[5] https://github.com/waylan/beautifulsoup/
[6] https://github.com/tsproisl/SoMaJo/
[7] https://github.com/nltk/nltk/
[8] https://github.com/explosion/spaCy/

need to reliably determine the language of a web page's content.

#### LangID

*LangID*[9] is capable of determining the language of a given text sequence. The LangID multinomial naive Bayes model is trained to determine a text's language among 97 (as of 20210703) languages including English and German [15]. In the context of this thesis LangID classifies text as either English or German but no other language.

#### Alternatives

Alternatives were not considered as LangID proofed to deliver reliable results. Training a naive Bayes model for the specific domain of T&Cs from scratch was not necessary either.

## 4.3. DOM-Tree

As Selenium, which is used for the HTML download, does not offer full XPath support, as one cannot address text element, i.e. all children of a DOM-node including text elements, there is a need to parse the downloaded HTML into a customized DOM structure. As described above, the XML parser lxml comes with full XPath support. In order to reduce overhead, the data structure resulting from the lxml HTML parser is transformed into a data structure much more fitting for this project (see Figure 4.3). This data structure also supports the integration of rendered CSS styles, i.e. only those properties of the style actually needed during hierarchy extraction.

All HTML tags not rendered out are not considered in the DOM-tree. To identify these HTML tags, all tags listed in the HTML-standard[10] were investigated.

The desired format becomes necessary for hierarchy extraction as it allows for a more precise analysis of the style of text sequences. As described in Section 4.5 the approach used for hierarchy extraction partly relies on information on the visual style of text sequences.

Limiting the style extraction to those elements which were previously identified as main content offers some room for optimization as the number of nodes for which the style needs to be extracted is far lower. This behavior is visualized in the sequence

[9] https://github.com/saffsd/langid.py
[10] https://html.spec.whatwg.org/

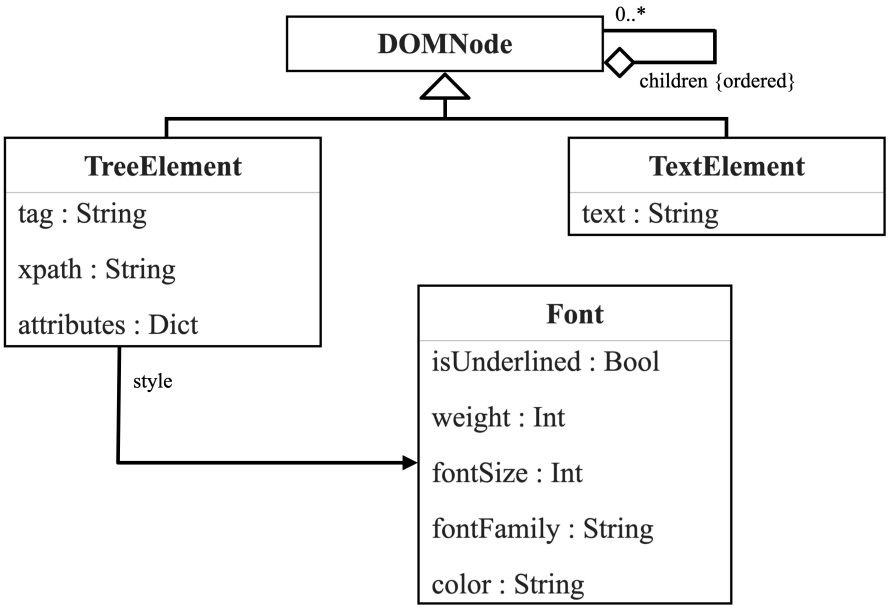



Figure 4.3.: UML diagram of data structure used for DOM-tree including CSS rendered style information.

diagram in Figure 4.2, where the style for the main content subtree is extracted if this had not happened before.

## 4.4. Content Extraction

As described in Chapter 3, the main content of a web page usually shares a common style. As the main content is, in the case of T&C pages, almost always the largest visible content block, the extraction approach is centered around the idea of identifying the style used for the main content. This can usually be done by counting the number of characters per style.

Styling is added to HTML using CSS which allows defining styles for certain nodes. This can also be extended by defining style classes and IDs which can be assigned to DOM-nodes as attributes. CSS style information can also be directly assigned as attributes to the DOM-nodes. To identify the Most Common Style (MCS), the library uses a depth-first search to traverse the DOM-tree. The style of a node and the number of characters subject to this styling are saved to a python dictionary.

Several approaches of approximating the style including their advantages and disad-

vantages are presented below.

Since some tags are not rendered by browsers, the dictionary is filtered for these tags to exclude them when determining the MCS.

After the MCS is identified, the tree is traversed to identify a node covering at least 85% (This threshold is variable. The reason for its selection can be found in Section 4.4.1.) of the characters of the MCS. Once this node is identified, all descendants are classified as content. This decision is justified by the findings in Chapter 3: T&Cs are usually continuous texts not interrupted by any form of advertisement or related content as it often is the case in newspaper articles. As most Content Management Systems (CMS) structure their main content using a container (e.g. `<div>`), it is sufficient to identify this container to also identify all of the main content. Headings or withdrawal forms, which usually follow a different style than the MCS, are also included in the main content node as its children.

In some rare cases CMSs do not use a container to hold the whole main content. Instead individual containers onto which the content is divided are placed as direct descendants to the `<body>` node. By identifying the longest subsequence containing the MCS, one can most likely extract the main content while excluding boilerplate content like navigation bar and footer. This fallback solution provides much worse results than the actual content extraction algorithm.

**Algorithm 1:** Extract main content by common ancestor(s).

```
Input: DOM-node holding the HTML body, threshold
Result: DOM-node holding the main content
mostCommonStyle ← getMostCommonStyle(bodyList);
sinlgeCommonAncestor ← findLowestCommonAncestor(bodyList,
 mostCommonStyle, threshold);
if singleCommonAncestor ≠ None then
    /* singleCommonAncestor exists and is returned                    */
    return singleCommonAncestor;
else
    /* no surrounding container, all content nodes direct descendants of
       the body node                                                   */
    ancestorSequence ← getMaximalSubsequenceOfStyle(bodyList,
     mostCommonStyle);
    body.children ← ancestorSequence;
    return body;
end
```

### 4.4.1. Threshold

The selection of the correct standard threshold is a crucial step in the implementation of the extraction algorithm. A threshold too high can result in an extraction failure and trigger the fallback solution described above. A threshold too low can result in extracting only parts of the content. In addition, a threshold must be above 50% as a lower threshold would result in multiple possibilities for selecting the main content node.

In order to identify the correct threshold, the sample used in Chapter 3 is used again. As described in Appendix A, the T&C pages from the German sample hold less boilerplate content in most of the cases. Therefore it is reasonable to analyze the effects of threshold selection for the German and English samples independent of each other. As the differences are unlikely to be due to the language area and more likely due to the size of the online shops, language-specific thresholds are not used in the implementation. The goal should be the identification of a threshold that produces good results on both samples.

The styles were determined using the *Common Ancestor Extractor: Naive Style and Short Text Exclusion* presented below. Applying the presented optimal threshold is thus limited to this extractor and the domain of T&Cs.

The meaning of the brackets used in Tables 4.1 and 4.2 is the following:

$$x \in [a; b) \mid x \geq a \land x < b$$

The threshold must be as low as possible to allow the algorithm some tolerance for nodes that contain text in the MCS but are not part of the main content. However, it must not be so low, that the algorithm might classify one of its children with lower coverage as the main content.

The lower bound of the interval of possible threshold values is limited to approximately 83.65% by the English sample, i.e. the lowest coverage of the *next largest coverage in child node*. The upper bound of this interval is defined by the lowest coverage of the *main content node* above 83.65%. It can be found in the English sample with a value of approximately 91.01%.

Selecting a threshold in this interval would trigger the fallback algorithm in two cases (the case in the German sample is ignored, as it cannot be avoided due to its general lack of a node with coverage greater than 50%) in the sample, as the actual main content node would cover less than 83.65% of the relevant characters. Selecting a lower threshold would not improve the extraction results, as the amount of additional correctly identified main content nodes is lower than those which would be misclassified

| Coverage | of main content node | of next largest coverage in child node |
|---|---|---|
| 1.0 | 23 | 0 |
| [0.95; 1.0) | 6 | 0 |
| [0.90; 0.95) | 0 | 0 |
| [0.85; 0.90) | 0 | 0 |
| [0.80; 0.85) | 0 | 0 |
| [0.75; 0.80) | 0 | 0 |
| [0.70; 0.75) | 0 | 1 |
| [0.65; 0.70) | 0 | 1 |
| [0.60; 0.65) | 0 | 2 |
| [0.55; 0.60) | 0 | 0 |
| [0.55; 0.50) | 0 | 0 |
| NONE - [0.0; 0.5) | 1 | 25 |

Table 4.1.: Coverage of the main content node by the total number of characters of the MCS and its child with the next largest coverage in the German sample.

additionally.

Thus the threshold is set to 85%.

### 4.4.2. Common Ancestor Extractor: Naive Style

The first more naive approach of approximating the styles of DOM-nodes is to combine the tag and its assigned attributes (<*tag*>${*attributes*}). This approach ignores the inherited CSS properties from parent nodes. Theoretically, style properties could be inherited from a node's ancestors. Even the style of two supposedly identical <p>${} elements (because they have the same attributes) can be rendered into a different visual style by the browser depending on their ancestors.

As this approach avoids rendering out the CSS styles it is rather efficient. The correct content was identified in the majority of the investigated cases from the sample (sample is presented in Appendix A). Some problems occurred on those sites which were using the same tag for their main content and other elements. If the amount of characters is low this does not lead to a problem as the threshold of 85% allows for a maximum of 15% of the same styled characters to be out of the suspected main content node. Whenever the amount of characters becomes too high, e.g. in navigation bars, the extractor needs to select a higher common ancestor - and thus more content than the actual main content. The actual styling of the nodes using the same tag and the

| Coverage | of main content node | of next largest coverage in child node |
|---|---|---|
| 1.0 | 11 | 0 |
| [0.95; 1.0) | 5 | 0 |
| [0.90; 0.95) | 2 | 0 |
| [0.85; 0.90) | 0 | 0 |
| [0.80; 0.85) | 0 | 1 |
| [0.75; 0.80) | 0 | 0 |
| [0.70; 0.75) | 1 | 0 |
| [0.65; 0.70) | 0 | 0 |
| [0.60; 0.65) | 0 | 0 |
| [0.55; 0.60) | 0 | 0 |
| [0.55; 0.50) | 1 | 0 |
| NONE - [0.0; 0.50) | 0 | 18 |

Table 4.2.: Coverage of the main content node by the total number of characters of the MCS and its child with the next largest coverage in the English sample.

same attributes could still be very different as a result of the nodes' ancestors and the attributes assigned to them - thus including the descendants' attributes or rendering out the actual style could fix this problem. Another possible approach to fix this issue is the exclusion of text elements that occur to be a headline or part of a navigation bar.

### 4.4.3. Common Ancestor Extractor: Rendered Style

Selenium allows for rendering out the actual style of a node (and its associated text sequence) allowing the library to determine the actual MCS. The style is defined by font-size, font-weight, underlining, font-family and font-color.

This approach needs to render the style of each node in the web pages HTML-body which results in significant overhead. The number of nodes in the German sample is shown in Figure 4.4, the number of nodes for the English sample is shown in Figure 4.5. Even though CSS style information will be needed for the hierarchy extraction, the amount of nodes requiring this can be reduced by determining the main content using the *Naive Style* approach before extracting the style of nodes. This would limit the style extraction to the relevant nodes. In addition, the style extraction for *TreeElement*s without any direct *TextElement* child nodes would not be necessary.

During a quick evaluation, the *Rendered Style* approach did perform no better than the *Naive Style* approach.

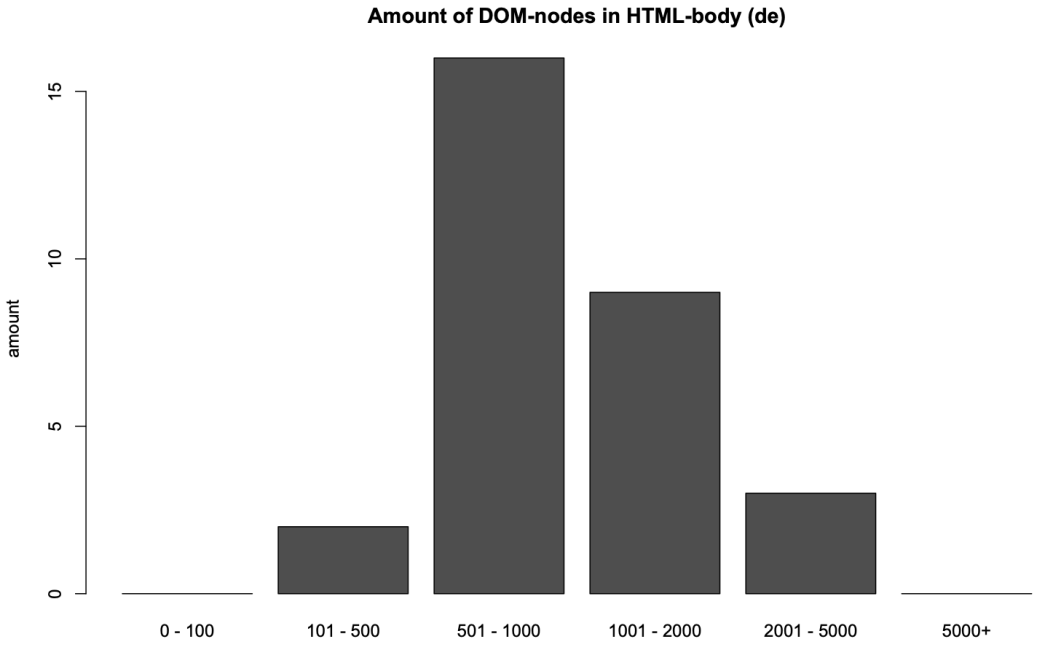



Figure 4.4.: Amount of DOM-nodes used in the body of T&C-pages in the German sample.

### 4.4.4. Common Ancestor Extractor: Naive Style and Short Text Exclusion

As short text elements like headlines and those that occur in navigation bars are usually not considered to be the main content of a T&C web page, DOM-nodes containing such short nodes can be ignored when determining the MCS. The *Naive/Rendered Style and Short Text Exclusion* implementation requires a minimum of four words in a DOM-node to account for it when determining the number of characters for each style. A word is considered to be a sequence of characters surrounded by whitespaces.

As *Naive Style* and *Rendered Style* approaches showed no significant differences in terms of the results, the approach including *Short Text Exclusion* is combined with the more efficient *Naive Style* implementation.

As style is only extracted for the nodes which are actually part of the main content, the number of nodes for which a style extraction is needed can be reduced by approximately 69% in the German sample (Figure 4.6 and by approximately 71% in the English sample in average after using the *Common Ancestor Extractor: Naive Style and Short Text*

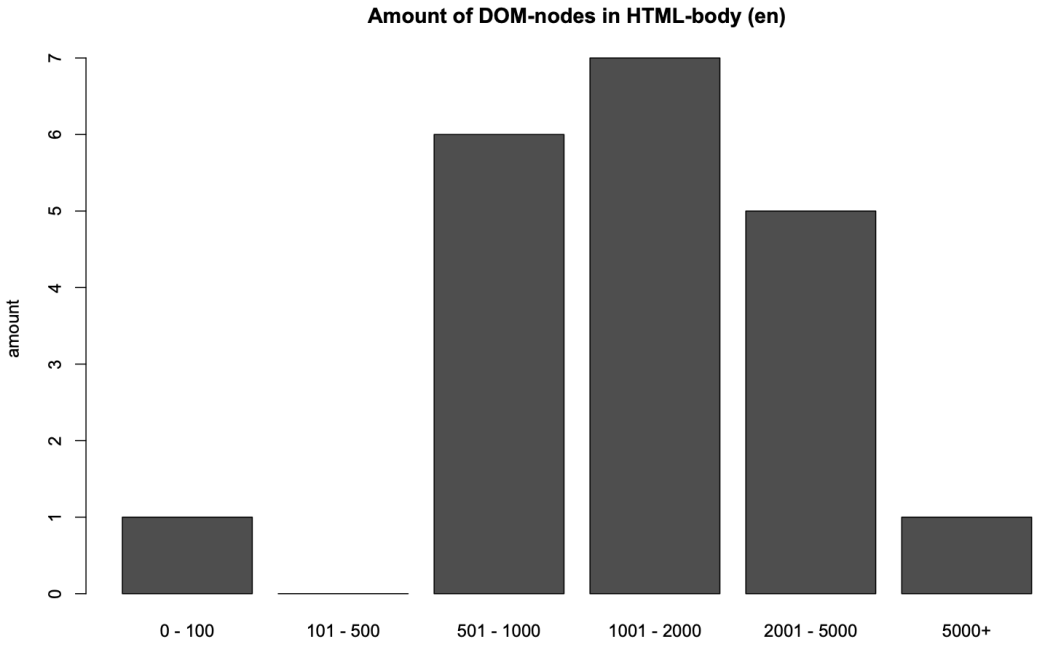



Figure 4.5.: Amount of DOM-nodes used in the body of T&C-pages in the English sample.

*Exclusion*. Due to the use of different CMSs by web pages investigated in the sample, there is some variation in the node reduction ranging from 7.5% to 99%.

## 4.5. Hierarchy Extraction

The hierarchy extraction is based on the idea presented by Manabe and Tajima [16]. The visual style of the text is used to identify headings and their associated text blocks. In some cases, T&Cs are also structured using enumerations or a combination of visual style and enumerations (see Figures 3.2 and 3.3). Thus, this information needs to be considered, too. For the actual hierarchy extraction information from both, the enumerations and the visual styles, need to be taken into account in a rule-based approach to produce accurate results.

Structural information from the DOM-tree is not used for hierarchy extraction as many web pages do not represent the hierarchical structure of the T&Cs in the DOM-tree. As

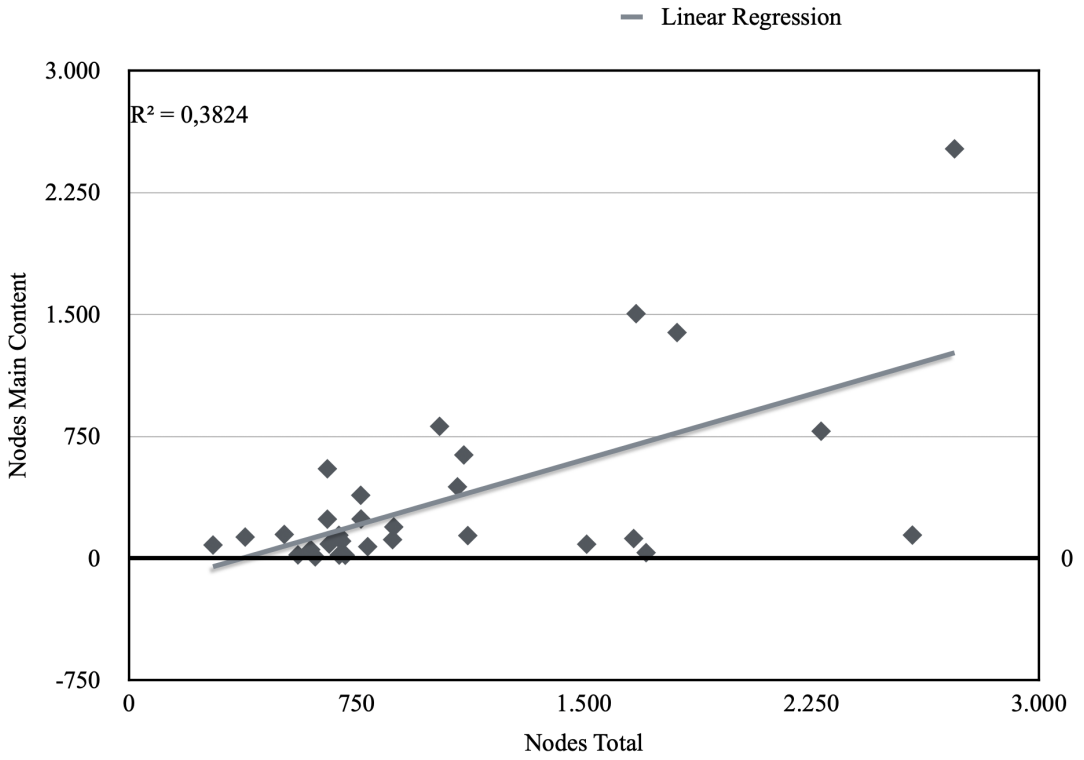



Figure 4.6.: Relationship of the total amount of DOM-nodes and the amount of DOM-nodes identified as part of the main content in the German sample.

discovered by Manabe and Tajima [16], the information from the DOM-tree is often misleading as e.g. `h`-tags are abused for SEO and many web pages are not even using special HTML-tags but CSS to create visual differences in the page content. Therefore it is justified to represent the main content as a continuous text (blocklist, see Section 4.5.1) with style information attached.

The algorithm used to structure the extracted content hierarchically uses the information on visual styles with a higher priority and relies on the parsed enumerations to verify and adjust the results. This decision was made based on the data presented in Figures 3.2 and 3.3 which provides evidence, that visual features are almost always part of the hierarchical structure of the investigated German and English T&C web pages.

Since most of the investigated pages also use enumeration patterns for hierarchical structuring, this information is also taken into account during hierarchy extraction.

An example structure of the desired results of the hierarchy extraction can be seen in

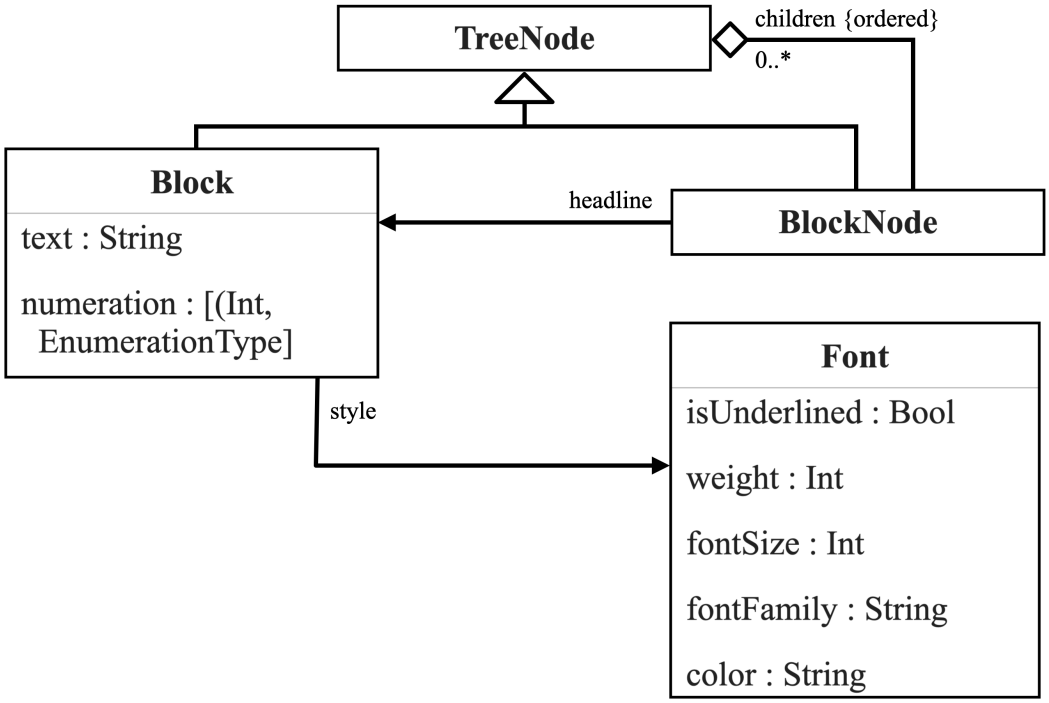



Figure 4.7.: UML diagram of the data structure used to represent the hierarchy tree.

Figure 4.8.

### 4.5.1. Content Blocks

As structural information arising from the DOM-tree is not regarded during hierarchy extraction, a different input format than the previously presented DOM-node (Figure 4.3) is used. The previously identified main content is now regarded as a list of content blocks where each block has its own style. A block is a sequence of characters ending with a forced newline. This forced newline could be a `<br>` tag, the start or end of a paragraph (`<p>`), etc. All HTML tags from the HTML-standard[12] were investigated in order to identify those forming new paragraphs.

This is similar to the approach presented by Viveros et al. [11].

The style of a block corresponds to the style that the majority of the characters in it are

[11]https://www.aldi.co.uk/customer-services/terms-and-conditions/ last accessed on 20210325
[12]https://html.spec.whatwg.org/

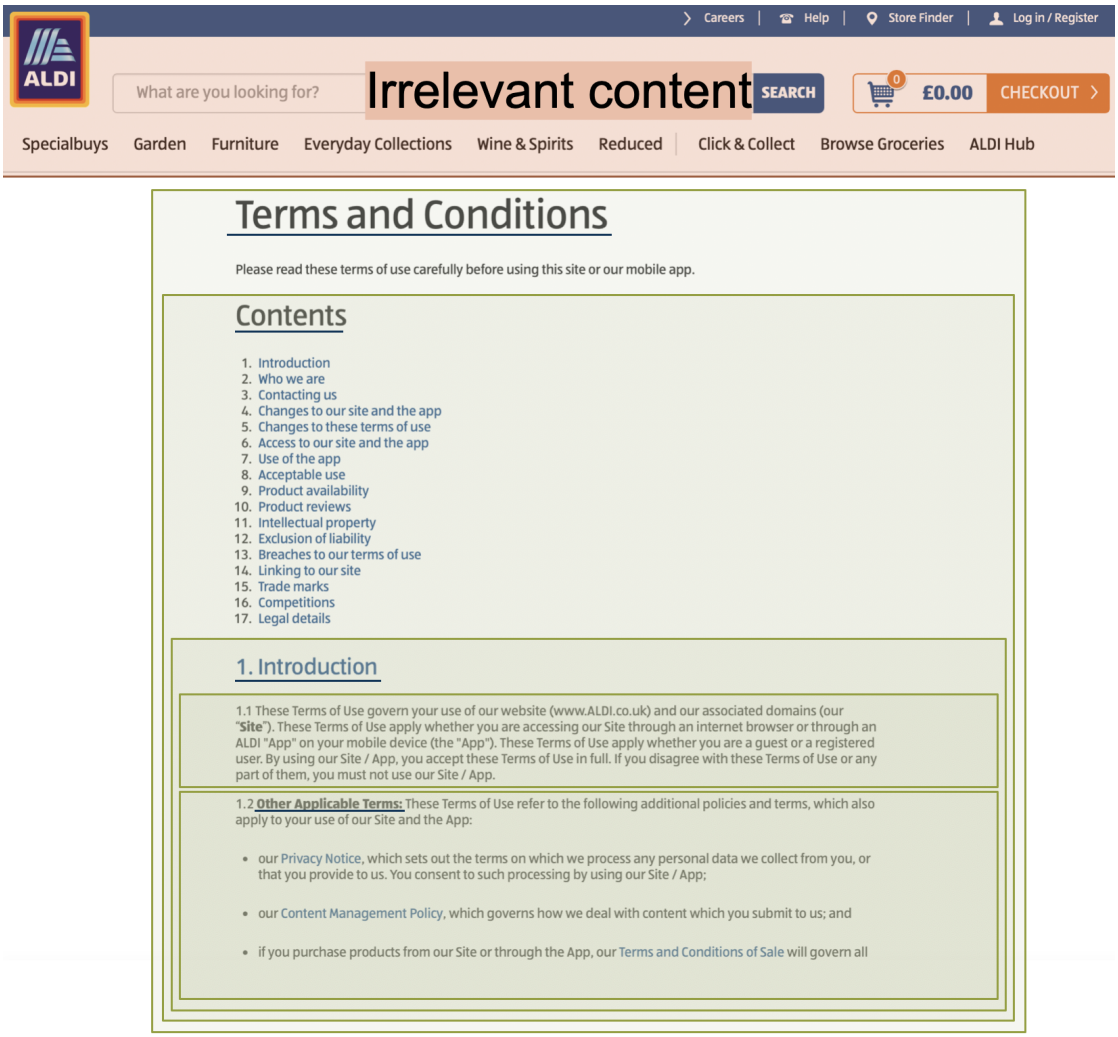



Figure 4.8.: Extracted Hierarchy of ALDI UK[11] T&C web page (segments in green boxes, irrelevant content in red boxes, headlines with blue underlining).

part of. This solves potential issues during hierarchy extraction which might arise from highlighted words in bold or underlined styling occurring in the middle of a regular content block. The style of content in anchor tags (<a>) is only considered as a fallback solution if no other style is visible in the block.

An example of block formation can be seen in Figure 4.9. Its second block's code is represented by the HTML code in Listing 4.1.

Listing 4.1: HTML representation of the second block in Figure 4.9.

```
<p>We do not disclose buyers' information to third parties other than
when order details are processed as part of the order fulfilment
process, e.g. courier companies. In this case, the third party will
not disclose any of the details to any other third party. To read
our full GDPR policy, please click
    <a href="https://www.telescopehouse.com/bresseruk-privacy-policy">
        <span style="text-decoration: underline;">
            <strong>here</strong>
        </span>
    </a>
.</p>
```


Figure 4.9.: Content blocks from Telescopehouse[13] visualized (blocks in green boxes). One non-paragraph forming DOM-node is aggregated to the paragraph forming the second DOM-node.

The extraction of potential enumeration patterns described in Section 4.5.3 is triggered automatically for each block.

List elements (`<li>`) receive their own special list enumeration pattern.

[13]https://www.telescopehouse.com/terms-and-conditions/ last accessed on 20210715

### 4.5.2. Visual Styles

Visual styles of the text are extracted as differences in weight, text-decoration, and size of the text. They are one of the common ways to structure a document and distinguish between headline and content. Selenium provides a method (`.value_of_css_property(<prop>)`) to retrieve selected CSS properties of DOM-nodes. These visual styles extracted by selenium are attached to the custom DOM-tree data structure presented in Figure 4.3.

The font size can be extracted using the attribute `font-size` which returns the height of the font in pixels. The font weight is extracted using the attribute `font-weight` returning the weight as an integer. Whether a text is underlined can be retrieved by the attribute `text-decoration` which has to contain the word *underline* in order to render a line underneath the text in the browser.

The following assumptions introduced by Manabe and Tajima [16] are used as a basis for the visual hierarchy extraction:

1. headings appear at the beginning of the corresponding blocks
2. headings are given prominent visual styles
3. headings of the same level share the same visual style

A section's start is identified by determining its headline, i.e. a line with a different style than the MCS identified during content extraction. A headline is only allowed a maximum of 10 words (character sequences separated by whitespace). The section ends whenever the next line styled like the current section's headline occurs. The content in between these two lines forms the provisionally content of the upper headline. Each of the identified sections is then grouped into subsections using the same algorithm (Algorithm 2) until no more prominent style is visible in between two headlines.

Content in between two headlines, respectively the last headline and the end of the main content is assumed to be the content of the upper headline.

### 4.5.3. Enumerations

Enumerations are extracted using a regular expression and the standard Python *re*[14] library. The regular expression supports the extraction of single-styled enumerations or combinations of different enumerations styles. The different numbering styles are brought into a uniform format so that they can be compared with each other. Each enumeration is brought to a uniform format (`int`) by the methods specified below. Enumerations are then represented as a list where each level of enumeration

[14] https://github.com/python/cpython/blob/3.9/Lib/re.py

**Algorithm 2:** Extract hierarchy based on headlines (recursive).

```
Input: List of blocks (blockList)
Result: Children of a Node
/* Determine headline style of current level and gather all headlines
   on this current level.                                              */
headlineStyle ← getNextHeadlineStyle(blockList);
headlineList ← [];
for block in blockList do
    if block.style = headlineStyle then
        headlineList.append(block);
    end
end
/* Create children list for current node by adding the blocks
   associated to the current node and by extracting the lower level
   nodes.                                                              */
children ← [];
children.append(blockList[0 : headlineList[0].index];
for headline in headlineList do
    cChildren ← extractHierarchy(blockList[(headline.index + 1) :
     headline.next.index];
    children.append(Node(headline, cChildren));
end
return children;
```

is saved to the list as another entry (tuple of `int` and enumeration type (numeric, Roman, alphabetic)). These representations of enumeration patterns are linked to blocks/paragraphs allowing another algorithm to sort the paragraphs into a tree structure.

The regular expression shown below is used to detect enumeration patterns in the first 10 characters of each block.

```
\s[\(§]?(([IVXLivxl]{1,7})|([0-9]{1,2})|[a-zA-Z])
([\.\-,:](([IVXLivxl]{1,7})|([0-9]{1,2})|[a-zA-Z]))*[\-:\.)]?\s
```

Extracted enumerations are used to correct and validate the existing hierarchy which was previously extracted using an approach based on visual features (see Section 4.5.2).

In the first step, the blocks assigned to the individual sections are examined for possible numerical hierarchies. After the first enumeration pattern from the previously extracted

enumeration is selected as the headline style, all headlines of the same style in the subsection are gathered in order to create further subsections similar to the algorithm used during the visual separation.

An enumeration pattern is only considered if there are at least two consecutive numberings of that pattern whose numerical values reflect a valid step. A *1.1* followed by a *1.2* is thus classified as a valid step whereas a *1.1* followed by a *1.7* is invalid. Invalid steps or enumeration patterns occurring only once are probably found due to an error in the enumeration detection.

There is some potential danger in working with enumerations, as some T&C pages preface their main content with a table of contents which itself is enumerated in most cases. An example is presented in Figure 3.4. This potential source of errors is dealt with by requiring each headline to have at least one following text block, a lower level headline, or a minimum length of 10 whitespace-separated words.

After the initial enumeration-based segmentation within the nodes' content blocks, all headlines on the same level of the tree are checked for enumeration patterns. If there are different enumeration patterns on one level, the tree is modified in order to have consistent enumeration.

#### Arabic enumeration

Arabic numbers are the most common enumeration used in the domain of T&C structuring. They can easily be extracted from a string and transformed into a numerical representation using the build-in `int()` function of Python.

#### Roman enumeration

Since in a few cases Roman numerals were also used for numbering, these must be converted into Arabic numerals. In all investigated cases Roman numerals were not bigger than 20. The implementation covers the number range from 1 to 88, so all other eventualities do not raise any problems. The number range can be extended as desired by expanding a dictionary used to resolve the Roman numeral characters to their Arabic decimal counterpart and by extending the regular expression used for detecting enumeration patterns. The algorithm developed to transform Roman numerals to Arabic numerals is shown below (Algorithm 3). It works by adding up the respective values of the Roman numeral characters and subtracts their value if a larger

Roman numeral character has already been processed before.

**Algorithm 3:** Convert Roman numerals to decimal.

```
Input: Roman numeral (as string)
Result: number (as integer)
reversed ← reverse(number);
indexList ← ['I', 'V', 'X', 'L'];
resolve ← { 'I' : 1, 'V' : 5, 'X' : 10, 'L' : 50 };
lastIndex ← 0;
sum ← 0;
for character in reversed do
    if indexList.indexOf(character) < lastIndex then
        sum ← sum − resolve[character];
    else
        lastIndex ← indexList.indexOf(character);
        sum ← sum + resolve[character];
    end
end
return sum;
```

#### Alphabetic enumeration

As alphabetic enumeration is only applicable for single characters, there is no need for extensive conversion. Letters are simply converted to their lowercase counterpart to allow the processing of either lower- or uppercase enumerations. Their numeric value is determined by reducing their *ASCII* value by 96. This maps `a` to 1, `b` to 2, and so on.

#### Lists

As described in Section 4.5.1, list elements (`<li>`) are automatically separated into blocks with their own enumeration pattern. This is necessary, as enumerations are often not represented in the text of DOM-nodes but rendered by the browser.

The list elements allow for one last adjustment to the hierarchy tree: All blocks with list enumeration patterns underneath the same headline are separated into subsections.

The assumption, that no section is interrupted by a subsection and continued afterward is relaxed for the list enumeration pattern. Content before and after the list blocks is packed into subsections on the same level as the list blocks. This decision is justified as

most of the lists are actually interrupting subsections which are continued afterward.

## 4.6. Target Format

The target format was predefined by the team of AGB-Check [6] (see Figure 4.10).

The *Document*s attributes are retrieved in the following ways: The *id* is generated by hashing the URL and a timestamp, the source URL is given as an input to the library, the extraction date is determined from the computer time using the Python *Date* library, and the title is taken from the header of the HTML-document (`<title>`).

The *Section*s are generated from the extracted content (see Section 4.4) and its hierarchy (see Section 4.5) based on the hierarchy tree presented in Figure 4.7.

The objects are dumped to JavaScript Object Notation (JSON) using the standard python JSON library[15].

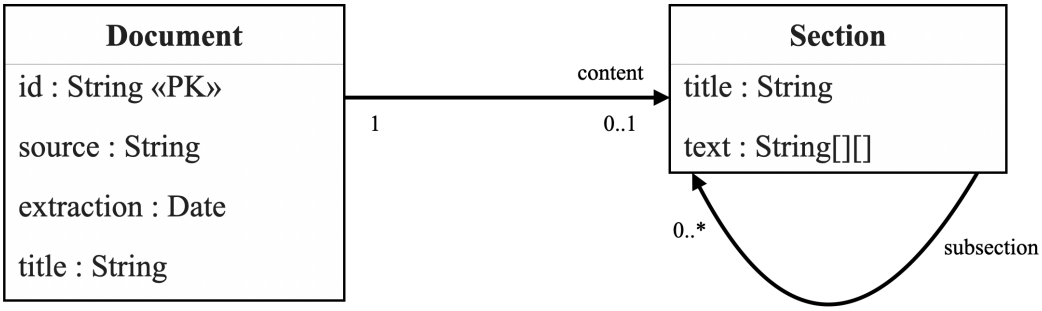



Figure 4.10.: Target format for the library's output.

[15]https://github.com/python/cpython/tree/3.9/Lib/json

# 5. Evaluation

The performance of the developed algorithms is evaluated in this chapter.

Besides the sample used to derive the requirements, there is also another sample to evaluate the library to prelude overfitting.

## 5.1. Content Extraction

The content extraction algorithm was tested with the sample used to derive requirements (see Appendix A) and another test sample solely created for the purpose of evaluation. Focus is on the correct identification of the start and end of the content, as the center is always identified correctly given the functionality of the previously introduced *LowestCommonAncestorExtractor* with *NaiveStyle*, *ShortTextExclusion*, and a threshold of 85%.

During the evaluation three main sources of errors were identified:

1. Threshold too high: Whenever the threshold is too high for the page (reasons for threshold selection are given in Section 4.4.1), the fallback algorithm is triggered.
2. No container for main content: Whenever the T&C page lacks a container wrapping the whole main content, the fallback algorithm is triggered.
3. Use of different tags: The *NaiveStyle* approach cannot handle the usage of different tags rendering to the same actual style used for the main content. If one of the tags is held in its own container, as it often ist the case with lists (`<ul>` or `<ol>`; text in `<li>`), only this container is extracted.

#### Requirements Sample

The results of the evaluation based on the sample used to derive requirements are presented in Tables 5.1 and 5.2.

The *LowestCommonAncestorExtractor* produced results in slightly better than those of *Trafilatura* which turned out to be the best library during the evaluation of existing

solutions. All errors except one are due to the use of the fall-back algorithm which is used whenever there is no node holding at least 85% of the characters of the MCS.

The other error, which cannot be blamed on this problem, resulted from the use of different tags for the main content. The web page with incorrect results used some paragraphs (`<p>`) and a list with its own sub-container (`<ul>`). Through this page layout, the list elements (`<li>`) were recognized as MCS and only the list was extracted.

The T&C page of ID 141 in the German sample (Table A.1) is built in a way, that made all of the existing solutions fail. The *LowestCommonAncestorExtractor* was able to extract the main content correctly.

| | too late | too early | correct |
|---|---|---|---|
| **LowestCommonAncestorExtractor (see Chapter 4)** | **1** | **3** | **45** |
| Boilerpipe ArticleExtractor | 16 | 4 | 24 |
| Boilerpipe LargestContentExtractor | 29 | 2 | 13 |
| Boilerpipe CanolaExtractor | 7 | 15 | 22 |
| JusText | 6 | 11 | 25 |
| Trafilatura | 5 | 2 | 37 |

Table 5.1.: Performance of detecting the start of the T&Cs for the implementation presented in Chapter 4 and for each of the previously examined extractors (taken from Chapter 5). *LowestCommonAncestorExtractor* was able to process 49 out of 50 web pages, the other extractors processed 44 of 50 web pages from the requirements sample.

| | too late | too early | correct |
|---|---|---|---|
| **LowestCommonAncestorExtractor (see Chapter 4)** | **3** | **2** | **44** |
| Boilerpipe ArticleExtractor | 23 | 13 | 8 |
| Boilerpipe LargestContentExtractor | 32 | 1 | 11 |
| Boilerpipe CanolaExtractor | 3 | 27 | 14 |
| JusText | 8 | 15 | 19 |
| Trafilatura | 5 | 4 | 35 |

Table 5.2.: Performance of detecting the end of the T&Cs for the implementation presented in Chapter 4 and for each of the previously examined extractors (taken from Chapter 5). *LowestCommonAncestorExtractor* was able to process 49 out of 50 web pages, the other extractors processed 44 of 50 web pages from the requirements sample.

### Test Sample

For evaluation purposes, another sample of 30 German and 20 English T&C pages from the data set by Braun and Matthes [5] (GitHub[1]) is used. The data from the English sample was again of lower quality. 35% of the data did not refer to the correct T&C page of the respective online shop. These links were corrected manually.

The results of the evaluation based on the test sample are presented in Table 5.3.

| | too late | too early | correct |
|---|---|---|---|
| LowestCommonAncestorExtractor (see Chapter 4) - start | 3 | 0 | 46 |
| LowestCommonAncestorExtractor (see chapter 4) - end | 0 | 2 | 47 |

Table 5.3.: Performance of detecting the start and end of the T&Cs for the implementation presented in Chapter 4 based on the test sample. *LowestCommonAncestorExtractor* was able to process 49 out of 50 web pages.

The errors in the test sample can be traced to the same reasons as those identified in the requirements sample. The overall results from the test sample validate the results from the requirements sample. There is no statistical evidence of potential overfitting.

[1] https://github.com/sebischair/TC-Detection-Corpus

## 5.2. Hierarchy Extraction

The hierarchy extraction algorithm was, similar to the content extraction, evaluated with the sample used to derive requirements (see Appendix A) and a test sample created for the purpose of evaluation. Errors arising from a failed content extraction are ignored in this section, as they provide no information on the quality of the algorithm applied during hierarchy extraction.

The deviations of the hierarchy extraction algorithm from the expected results are determined by assigning the following scores to the extracted nodes:

- `0`: each section with correct parent, correct content, and correct title
- `0.4`: wrong parent
- `0.5`: wrong content
- `0.1`: wrong title

As different T&C pages contain different amounts of sections, the score is divided by the total amount of sections identified by the algorithm. An error score of 0 accounts for a perfect extraction.

As explained in the next paragraph, some properties of web pages can result in a large number of wrong sections, which the human user would not identify as such. The scores determined for the different T&C pages are thus only a rough estimate of the performance of the algorithm. A precise analysis of the sources of errors can be found in the next paragraph.

During the evaluation the following main sources of errors were identified:

1. Use of bold text: Some pages used bold text elements to highlight whole sections or just some blocks (see Section 4.5.1). Due to the algorithm's functionality, this can screw up the whole result as the bold blocks might be identified as headlines.
2. Wrong enumeration: A surprisingly large amount of T&C pages contains errors in their enumerations. As the algorithm requires a strict sequence of numerations, this can lead to problems in the hierarchy extraction.
3. Violation of the assumption "Sections are not interrupted": The algorithms assumes, that there is no more content of a section after one of its subsection, As this assumption is violated in some cases, this leads to errors in the extracted hierarchy.
4. Use of tables: Whenever tables occurred on a page (often in the context of shipping costs), the algorithm separated each cell into its own block resulting in a large number of blocks with different styles. A high frequency of numbers occurring

in the table worsened the results as the vast amount of detected enumeration patterns triggered further adjustments to the table.

5. Failed style extraction: In order to link the custom DOM-tree structure to the Selenium tree each DOM-node is attached its full XPath. As some pages render elements like a cookie banner after a short timespan, these elements may not be included in the parsed DOM-tree. As style extraction takes more time in the browser, new elements can render and tackle the validity of the XPath attached to the DOM-node. Visual features cannot be extracted resulting in an erroneous result during the visual-based hierarchy extraction.

In order to evaluate the hierarchy extraction algorithm a small console application was developed to support annotation of the algorithms results. It allowed the user to classify the content as either *correct* `xor` [*wrong parent* `or` *wrong content* `or` *wrong title*]. A random sample of 10% of the evaluated pages was annotated by a second person. No significant deviations from the first trial were found.

There is no statistical evidence of potential overfitting by comparing the error scores of the requirements- and test-sample.

The time consumed to extract the 100 pages was approximately 15 minutes, as the extraction of the DOM-nodes' styles is rather time-intensive.

The meaning of the brackets used in Tables 5.4, 5.5, 5.6, and 5.7 is the following:

$$x \in [a; b) \mid x \geq a \wedge x < b$$

**Requirements Sample**

The results of the evaluation based on the sample used to derive requirements are presented in Tables 5.4 and 5.5.

The larger number of failures in the English requirements sample mostly resulted from a failed content extraction. The 6 failed hierarchy extractions correspond to the 6 failed content extractions mentioned in Section 5.1.

| Error Score | Amount |
|---|---|
| 0 | 12 |
| (0; 0.05] | 12 |
| (0.05; 0.1] | 1 |
| (0.1; 0.15] | 2 |
| (0.15; 0.2] | 1 |
| (0.2; 0.3] | 1 |
| (0.3; 0.5] | 1 |
| (0.5; 1] | 0 |
| Failed | 0 |

Table 5.4.: Distribution of error scores for the hierarchy extraction of the German requirements sample.

| Error Score | Amount |
|---|---|
| 0 | 5 |
| (0; 0.05] | 4 |
| (0.05; 0.1] | 1 |
| (0.1; 0.15] | 2 |
| (0.15; 0.2] | 0 |
| (0.2; 0.3] | 0 |
| (0.3; 0.5] | 1 |
| (0.5; 1] | 0 |
| Failed | 6 |

Table 5.5.: Distribution of error scores for the hierarchy extraction of the English requirements sample.

### Test Sample

The results of the evaluation based on the test sample are presented in the Tables 5.6 and 5.7.

| Error Score | Amount |
|---|---|
| 0 | 8 |
| (0; 0.05] | 11 |
| (0.05; 0.1] | 1 |
| (0.1; 0.15] | 1 |
| (0.15; 0.2] | 1 |
| (0.2; 0.3] | 1 |
| (0.3; 0.5] | 5 |
| (0.5; 1] | 0 |
| Failed | 2 |

Table 5.6.: Distribution of error scores for the hierarchy extraction of the German test sample.

| Error Score | Amount |
|---|---|
| 0 | 9 |
| (0; 0.05] | 3 |
| (0.05; 0.1] | 2 |
| (0.1; 0.15] | 1 |
| (0.15; 0.2] | 2 |
| (0.2; 0.3] | 2 |
| (0.3; 0.5] | 1 |
| (0.5; 1] | 0 |
| Failed | 0 |

Table 5.7.: Distribution of error scores for the hierarchy extraction of the English test sample.

# 6. Conclusion and Future Work

In this chapter, the results are summarised. Furthermore, the topics that require further research are discussed.

In this thesis, a new content extraction algorithm was developed that performs better than existing solutions in its specific domain of T&C web pages. Since the algorithm is based on some domain-specific assumptions, it is unclear how successfully it would operate in a generic web corpus. Further research in the field could answer this question. Initial small tests looked promising under the assumption that the content of the page is not interrupted.

The style extraction is currently based on Selenium. However, the performance of retrieving certain CSS properties is rather low slowing down the whole algorithm. In the future, a more efficient solution for the extraction of styles may remedy this. Alternatively, several content extraction threads can operate in parallel.

The general functionality of the rule-based approach to hierarchy extraction could be demonstrated. The general idea of Manabe and Tajima [16] was extended by an enumeration detection due to the frequent usage of enumerations to structure T&C pages. It is much more difficult to achieve similar success rates in hierarchy extraction as with content extraction, due to the many irregularities in visual representation. This is probably due to the fact that the operators of different online shops often want to highlight very different elements from the contract text visually. In cases where such outliers do not occur in the visual representation, hierarchy extraction yields good results.

Accurate results are necessary in the legal context of AGB-Check as erroneous results propagate through the whole NLP-pipe.

**RQ1: What are special requirements for the extraction of contracts in comparison with *regular* (i.e. news articles, blogposts) content?**

The main difference in the properties of T&C pages compared to the *regular* ones found in this thesis was the continuous content. Compared to *regular* content, in which addresses and telephone numbers rarely appear and often indicate a footer or the

imprint, this information is part of the relevant content in the case of contracts.

**RQ2: How to extract the relevant parts from contracts in raw HTML?**

Content is extracted using the *LowestCommonAncestorExtractor* developed in this thesis. It is based on the idea of identifying a node holding a minimum of 85% of the MCS.

**RQ3: How to extract the structure and the hierarchy of paragraphs, (sub-)titles, and related clauses?**

The hierarchy is extracted using a rule-based approach. Structural information from the DOM-tree are only regarded in terms of paragraphs and list elements. The segmenting algorithm looks into the visual representation and enumerations attached to the paragraphs/blocks.

**StructuredLegalExtraction Library**

The methods provided by the library implemented based on the knowledge gained from *RQ2* and *RQ3* are documented in Appendix C.

# A. Sample

The sample used for the identification of requirements (see Chapter 3), the evaluation of different implementation approaches of the content extraction (see Section 4.4), and as a first try at the evaluation (see Chapter 5) was taken from the data set by Braun and Matthes [5] available from Github[1] under CC BY-SA 3.0 license. The sample was generated using the statistical software R[2].

### German Sample

The T&C pages of the German sample appear to be less overloaded than these of the English sample. This could be due to the smaller size of most of the online shops in the German sample.

| ID | URL |
|---|---|
| 4840 | https://www.uhrenbay.com/agb |
| 4026 | https://www.surifrey.com/agb |
| 4353 | https://www.benzkosmetik.de/agb |
| 2260 | https://campingtoilette-guenstig.de/allgemeine-geschaeftsbedingungen |
| 174 | https://steinehelden.de/agb |
| 3746 | https://corporate.brax.com/de_de/shop/agb-widerrufsrecht |
| 2899 | https://www.lederhose.com/de-AT/info/agb |
| 2533 | https://www.allergiker-shop-alfda.de/info/allgemeine-geschaeftsbedingungen.html |
| 3377 | https://www.radarfriends.net/agb/ |
| 4716 | https://www.my-perfect-skin.de/allgemeine-geschaeftsbedingungen |
| 1006 | https://www.handwerkerstore24.de/agb |
| 4515 | https://www.buegelrevolution.de/agb |
| 2791 | https://www.buyandfeelgood.de/de/unsere-agb |
| 4271 | https://www.metro.de/unternehmen/agb_marktplatz |
| 2853 | http://www.t-zone24.de/Unsere-AGBs.html |

[1] https://github.com/sebischair/TC-Detection-Corpus
[2] https://www.r-project.org

| ID | URL |
|---|---|
| 414 | https://www.pnshop.de/cgi-bin/pnshop/de_DE/generalTermsConditions.html |
| 2264 | https://www.augenblicke-eingefangen.de/agb-und-kundeninformationen |
| 2265 | https://www.tennis-heine.de/allgemeine-geschaeftsbedingungen |
| 3909 | https://www.schulranzen-fachmarkt.de/de/c/agb_1 |
| 3980 | https://www.wohnplanet.de/agb |
| 4678 | https://www.lohmeier-shop.de/agb |
| 3168 | https://www.parfuemerie-ruthe.de/AGB/ |
| 1465 | https://www.myspirits.eu/agb |
| 3879 | https://www.matratzen-concord.de/ueber-uns/agb/ |
| 4559 | https://www.empinio24.de/agb?belboon=2004111812239900144&pid=12401 |
| 3314 | https://somnishop.com/allgemeine-geschaeftsbedingungen/ |
| 1328 | https://www.stormbreaker.de/AGB |
| 3649 | https://www.spielzeug-und-mehr.de/AGB/ |
| 4384 | https://www.takeashot.de/pages/agb |
| 1458 | https://www.mein-gartenshop24.de:443/agb |

Table A.1.: German sample.

### English Sample

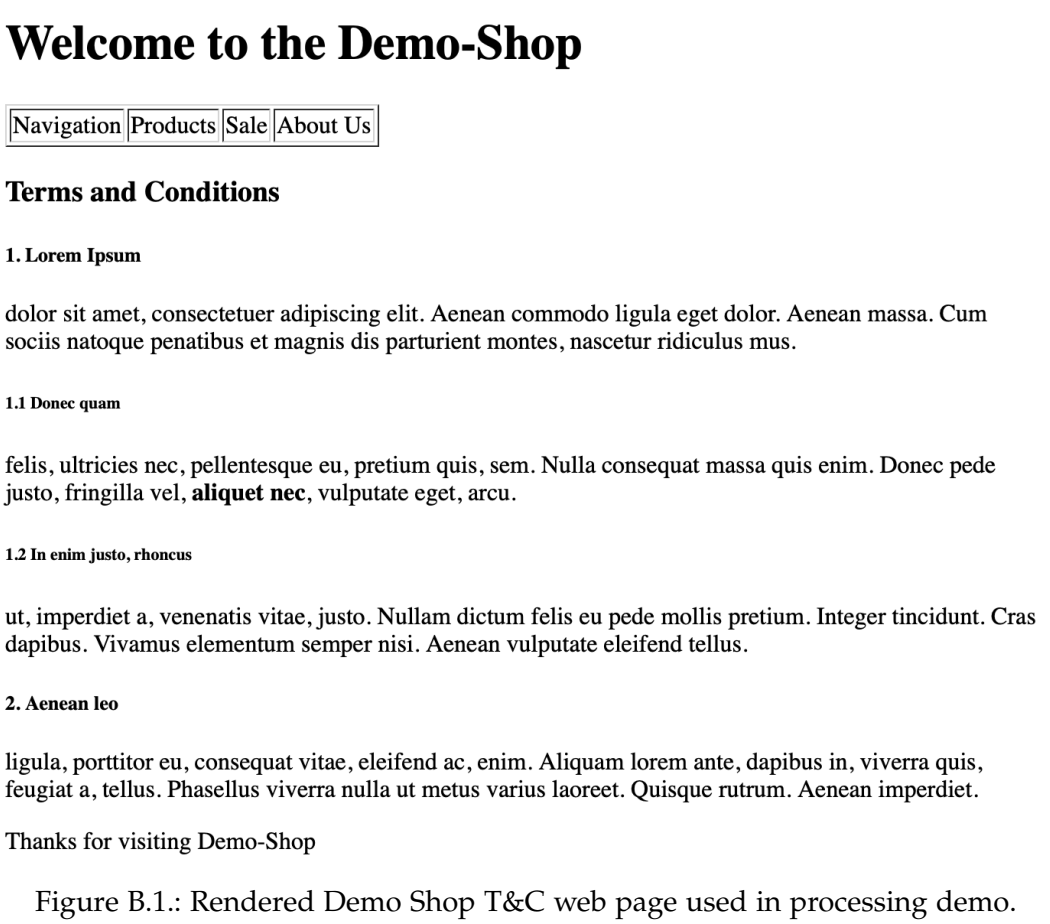
The T&C pages of the English sample do often contain more boilerplate content than those of the German sample. One reason could be the larger size of the online shops in the English sample.

| ID | URL |
|---|---|
| 7 | https://www.tyres-guru.co.uk/AGBs.html |
| 362 | https://www.ghostbikes.com/terms-and-conditions.html |
| 335 | https://www.electricalworld.com/en/Terms-and-Conditions/cc-3.aspx |
| 300 | https://www.forzasupplements.co.uk/pages/terms |
| 266 | https://www.telescopehouse.com/terms-and-conditions |
| 412 | https://www.chemist.co.uk/terms-and-conditions.html |
| 238 | https://www.samueljohnston.com/de/ABOUT—Terms-and-Conditions/cc-4.aspx |
| 99 | https://www.expresschemist.co.uk/info/terms-conditions |
| 126 | https://www.uktights.com/page/terms |
| 259 | https://www.gardenchic.co.uk/terms-conditions-i7 |
| 41 | http://www.cartridgepeople.com/UserTemplate/Content___page=terms.html |
| 173 | https://www.hunterboots.com/us/en_us/terms-of-sale |
| 417 | https://www.beautyflash.co.uk/terms-of-use.html |
| 81 | https://www.escentual.com/help-and-advice/terms-and-conditions/ |
| 37 | https://www.wexphotovideo.com/help/terms-and-conditions/ |
| 487 | https://www.just-rackets.co.uk/shop/customer-services.html#terms |
| 58 | https://www.jdsports.co.uk/customer-service/terms/ |
| 529 | https://thesportsedit.com/pages/terms-conditions |
| 345 | https://www.shoesyouwant.com/terms-conditions |
| 442 | https://www.dorothyperkins.com/page/terms-and-conditions.html |

Table A.2.: English sample, rows highlighted in red were updated as the link pointed to a wrong page.

# B. Processing Demo

This chapter presents the algorithm and the intermediary results in detail to allow for a better understanding of its functionality. Tasks fulfilled by other libraries or more generic tasks like parsing and downloading are not presented. The HTML code of the demo website used is presented in Listing B.1, a rendered version can be seen in Figure B.1.



## Welcome to the Demo-Shop

Navigation | Products | Sale | About Us

### Terms and Conditions

**1. Lorem Ipsum**

dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

**1.1 Donec quam**

felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, **aliquet nec**, vulputate eget, arcu.

**1.2 In enim justo, rhoncus**

ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus.

**2. Aenean leo**

ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet.

Thanks for visiting Demo-Shop

Figure B.1.: Rendered Demo Shop T&C web page used in processing demo.

Listing B.1: HTML representation of Demo Shop used in processing demo.

```
<html>
    <head>
        <title>Terms and Conditions of Demo-Shop</title>
    </head>
    <body>
        <div>
            <h1>Welcome to the Demo-Shop</h1>
            <table border='solid'>
                <tbody><tr>
                        <td>Navigation</td><td>Products</td>
                        <td>Sale</td><td>About Us</td>
                </tr></tbody>
            </table>
        </div>
        <div>
            <h3>Terms and Conditions</h3>
            <h5>1. Lorem Ipsum</h6>
            <p>dolor sit amet, consectetuer adipiscing elit. Aenean commodo
            ligula eget dolor. Aenean massa. Cum sociis natoque penatibus
            et magnis dis parturient montes, nascetur ridiculus mus.</p>
            <h6>1.1 Donec quam</h6>
            <p>felis, ultricies nec, pellentesque eu, pretium quis, sem.
            Nulla consequat massa quis enim. Donec pede justo, fringilla
            vel, <b>aliquet nec</b>, vulputate eget, arcu.</p>
            <h6>1.2 In enim justo, rhoncus</h6>
            <p>ut, imperdiet a, venenatis vitae, justo. Nullam dictum
            felis eu pede mollis pretium. Integer tincidunt. Cras
            dapibus. Vivamus elementum semper nisi. Aenean vulputate
            eleifend tellus.</p>
            <h5>2. Aenean leo</h5>
            <p>ligula, porttitor eu, consequat vitae, eleifend ac, enim.
            Aliquam lorem ante, dapibus in, viverra quis, feugiat a,
            tellus. Phasellus viverra nulla ut metus varius laoreet.
            Quisque rutrum. Aenean imperdiet.</p>
        </div>
        <p>Thanks for visiting Demo-Shop</p>
    </body>
</html>
```

## B.1. Content Extraction

The content extraction presented in this demo is using the *4.4.4 Common Ancestor Extractor: Naive Style and Short Text Exclusion*.

### Gathering the MCS (naive + min. of 4 words)

| Style | Characters | Characters discarded (less than 4 words) |
|---|---|---|
| `h1${}` | 24 | 0 |
| `td${}` | 30 | 30 |
| `h3${}` | 20 | 20 |
| `h5${}` | 27 | 27 |
| `p${}` | 742 | 0 |
| `h6${}` | 40 | 14 |
| `b${}` | 11 | 11 |

Table B.1.: Styles and their number of characters (naive + min. of four words). MCS highlighted in green, characters violating the *min. of 4 words* rule in red.

The style `p${}` is identified as the MCS. `p${}` described all dom nodes with tag `p` and not attributes assigned. As this is the *naive style* approach, this is only an approximation of the actual style.

The other two styles considered in the selection (`h1${}` [24 relevant characters] and `h6${}` [26 relevant characters]) are by far less prominent than `p${}`.

### Finding Lowest Common Ancestor covering 95% of `p${}` characters

Only DOM-nodes containing the MCS (`p${}`) are regarded in this step. The algorithm looks for the node with the highest depth and coverage above 95%. A list of all investigated nodes and the one covering at least 95% of the MCS is presented in Table B.2.

The threshold is set to 95% for *demonstration purposes* only. The total amount of characters of the demo shop is untypical low for a regular T&C web page justifying this adjustment in this particular case.

The nodes `/html/body/div[2]/p[`*1 - 4*`]` are investigated first, as their depth is higher than the depth of `/html/body/div[2]` and `/html/body/p`. As none of them is able

| Node (XPath) | Coverage | Depth |
|---|---|---|
| /html/body/div[2] | 0.9663573085846868 | 1 |
| /html/body/div[2]/p[1] | 0.2354988399071926 | 2 |
| /html/body/div[2]/p[2] | 0.19837587006960558 | 2 |
| /html/body/div[2]/p[3] | 0.2529002320185615 | 2 |
| /html/body/div[2]/p[4] | 0.27958236658932717 | 2 |
| /html/body/p | 0.033642691415313224 | 1 |

Table B.2.: Nodes containing style p${} with their coverage and depth. Lowest common ancestor holding the identified main content in green.

to reach a coverage of more than 95%, the next lower depth (1) is investigated. The DOM-node with XPath /html/body/div[2] has a coverage of more than 95% and is thus identified as the element holding the main content of the web page. An HTML representation of the main content can be found in Listing B.2.

Listing B.2: HTML representation of the identified main content.

```
<div>
    <h3>Terms and Conditions</h3>
    <h5>1. Lorem Ipsum</h6>
    <p>dolor sit amet, consectetuer adipiscing elit. Aenean commodo
    ligula eget dolor. Aenean massa. Cum sociis natoque penatibus
    et magnis dis parturient montes, nascetur ridiculus mus.</p>
    <h6>1.1 Donec quam</h6>
    <p>felis, ultricies nec, pellentesque eu, pretium quis, sem.
    Nulla consequat massa quis enim. Donec pede justo, fringilla
    vel, <b>aliquet nec</b>, vulputate eget, arcu.</p>
    <h6>1.2 In enim justo, rhoncus</h6>
    <p>ut, imperdiet a, venenatis vitae, justo. Nullam dictum
    felis eu pede mollis pretium. Integer tincidunt. Cras
    dapibus. Vivamus elementum semper nisi. Aenean vulputate
    eleifend tellus.</p>
    <h5>2. Aenean leo</h5>
    <p>ligula, porttitor eu, consequat vitae, eleifend ac, enim.
    Aliquam lorem ante, dapibus in, viverra quis, feugiat a,
    tellus. Phasellus viverra nulla ut metus varius laoreet.
    Quisque rutrum. Aenean imperdiet.</p>
</div>
```

## B.2. Hierarchy Extraction

The hierarchy extraction is based on the visual features and enumeration patterns in the main content.

### Creating the BlockList

As structural information is no longer regarded in this processing step, the DOM-tree is parsed into a *BlockList*. The section presented in Listing B.3 is subject to the merger of different DOM-nodes, as the <b> tag does not form its own block (i.e. paragraph) but is part of its parent <p> DOM-node. As the paragraph (<p>) contains 145 characters whereas its child (<b>) holds only 11 characters, the style of the parent (<p>) is adopted for the block.

Listing B.3: HTML representation of the paragraph <p> and its child <b> which need to be merged..

```
<p>felis, ultricies nec, pellentesque eu, pretium quis, sem.
Nulla consequat massa quis enim. Donec pede justo, fringilla
vel, <b>aliquet nec</b>, vulputate eget, arcu.</p>
```

The extracted *BlockList* is shown in Table B.3.

| ID | Text Content | Font-Size | Bold | Underlined |
|---|---|---|---|---|
| 1 | Terms and Conditions | 18px | yes | no |
| 2 | 1. Lorem Ipsum | 13px | yes | no |
| 3 | dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa.Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. | 16px | no | no |
| 4 | 1.1 Donec quam | 10px | yes | no |
| 5 | felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. | 16px | no | no |
| 6 | 1.2 In enim justo, rhoncus | 10px | yes | no |
| 7 | ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. | 16px | no | no |
| 8 | 2. Aenean leo | 13px | yes | no |
| 9 | ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. | 16px | no | no |

Table B.3.: BlockList generated from the previously identified main content.

### Visual Separation

The BlockList is now transformed into a tree build from nodes containing the previously formed blocks. The BlockList is separated based on the extracted style. The algorithm needs knowledge on the MCS which is determined the same way as during content extraction. The results of determining the MCS are shown in Table B.4.

| Style | Characters | Characters discarded (less than 3 words) |
|---|---|---|
| 18px, bold | 20 | 20 |
| 13px, bold | 27 | 27 |
| 16px | 724 | 0 |
| 10px, bold | 40 | 14 |

Table B.4.: Styles and their number of characters. MCS highlighted in green, characters violating the *min. of 3 words* rule in red.

The style with a font-size of 16px and no further properties is identified as the MCS for the visual separation. Thus all blocks with a different style are considered to be headlines.

Figures B.2, B.3, and B.4 visualize the steps of the separation algorithm.

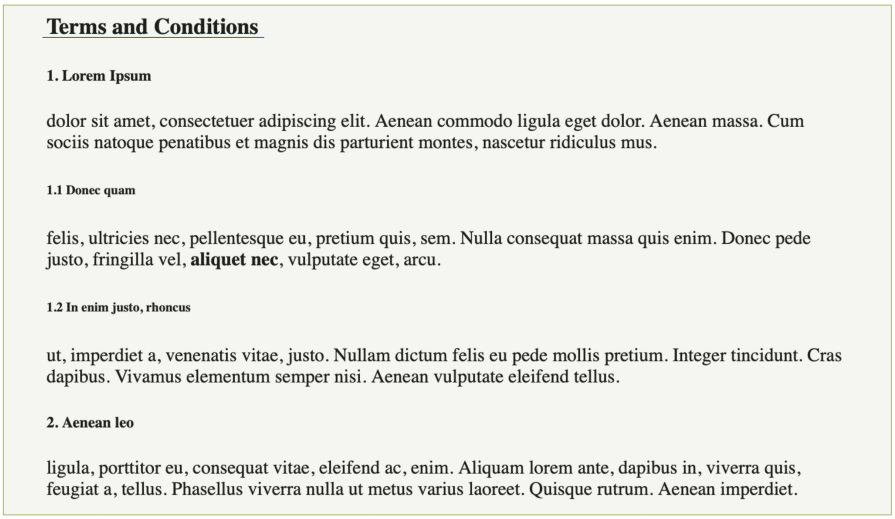



Figure B.2.: Step 1 of the visual segmentation algorithm. Segments in green boxes, headlines underlined in dark blue.

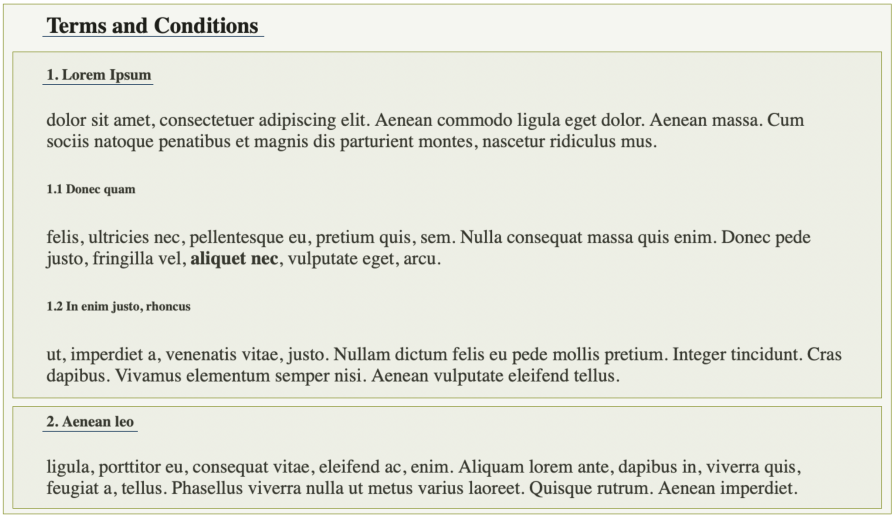



Figure B.3.: Step 2 of the visual segmentation algorithm. Segments in green boxes, headlines underlined in dark blue.

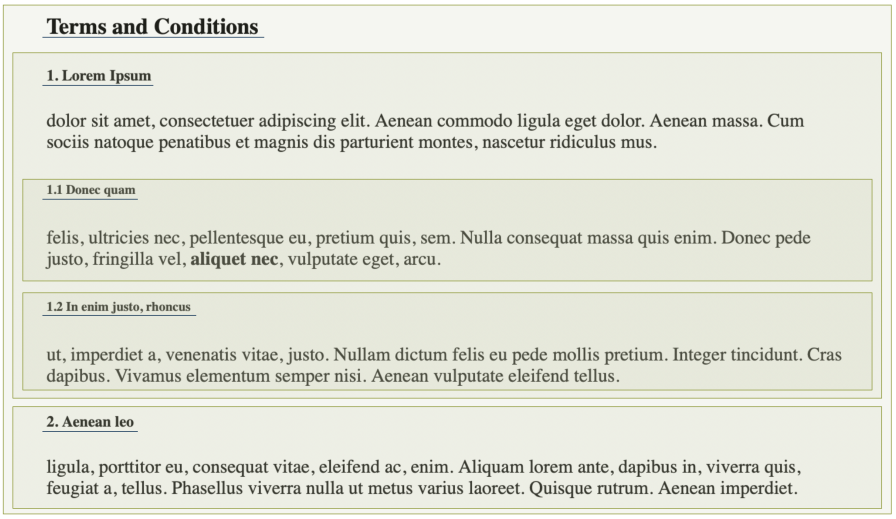



Figure B.4.: Step 3 of the visual segmentation algorithm. Segments in green boxes, headlines underlined in dark blue.

**Enumeration Patterns**

In order to adjust and validate the previous results, enumeration patterns of all blocks are examined. The results are presented in Table B.5.

| ID | Enumeration |
|---|---|
| 1 | [] |
| 2 | [1] |
| 3 | [] |
| 4 | [1, 1] |
| 5 | [] |
| 6 | [1, 2] |
| 7 | [] |
| 8 | [2] |
| 9 | [] |

Table B.5.: Enumeration patterns extracted for the previously specified blocks. IDs taken from Table B.3.

Thus, there are no further necessary adjustments to the identified enumeration patterns.

## B.3. Generating the Target Format

After identifying the language of the main content and separating the text in the non-headline blocks, the result is dumped into JSON format. The expected output for this demo process can be seen in Listing B.4.

Listing B.4: JSON result of the extraction.


```
{
{
    "content": [
        {
            "subsections": [
                {
                    "subsections": [
                        {
                            "subsections": [],
                            "text": [
                                [
                                    "felis",
                                    ",",
                                    "ultricies",
                                    "nec",
                                    ",",
                                    "pellentesque",
                                    "eu",
                                    ",",
```

```
                    "pretium",
                    "quis",
                    ",",
                    "sem",
                    "."
                ],
                [
                    "Nulla",
                    "consequat",
                    "massa",
                    "quis",
                    "enim",
                    "."
                ],
                [
                    "Donec",
                    "pede",
                    "justo",
                    ",",
                    "fringilla",
                    "vel",
                    ",",
                    "aliquet",
                    "nec",
                    ",",
                    "vulputate",
                    "eget",
                    ",",
                    "arcu",
                    "."
                ]
            ],
            "title": "1.1 Donec quam"
        },
        {
            "subsections": [],
            "text": [
                [
                    "ut",
                    ",",
                    "imperdiet",
                    "a",
                    ",",
                    "venenatis",
                    "vitae",
                    ",",
                    "justo",
                    "."
                ],
                [
                    "Nullam",
                    "dictumfelis",
                    "eu",
                    "pede",
                    "mollis",
                    "pretium",
                    "."
                ],
                [
                    "Integer",
                    "tincidunt",
                    "."
                ],
                [
                    "Cras",
                    "dapibus",
                    "."
                ],
                [
                    "Vivamus",
                    "elementum",
                    "semper",
                    "nisi",
                    "."
                ],
```

```
                    [
                        "Aenean",
                        "vulputate",
                        "eleifend",
                        "tellus",
                        "."
                    ]
                ],
                "title": "1.2 In enim justo, rhoncus"
            }
        ],
        "text": [
            [
                "dolor",
                "sit",
                "amet",
                ",",
                "consectetuer",
                "adipiscing",
                "elit",
                "."
            ],
            [
                "Aenean",
                "commodo",
                "ligula",
                "eget",
                "dolor",
                "."
            ],
            [
                "Aenean",
                "massa",
                "."
            ],
            [
                "Cum",
                "sociis",
                "natoque",
                "penatibus",
                "et",
                "magnis",
                "dis",
                "parturient",
                "montes",
                ",",
                "nascetur",
                "ridiculus",
                "mus."
            ]
        ],
        "title": "1. Lorem Ipsum"
    },
    {
        "subsections": [],
        "text": [
            [
                "ligula",
                ",",
                "porttitor",
                "eu",
                ",",
                "consequat",
                "vitae",
                ",",
                "eleifend",
                "ac",
                ",",
                "enim",
                "."
            ],
            [
                "Aliquam",
                "lorem",
                "ante",
```

```
                                ",",
                                "dapibus",
                                "in",
                                ",",
                                "viverra",
                                "quis",
                                ",",
                                "feugiat",
                                "a",
                                ",",
                                "tellus",
                                "."
                            ],
                            [
                                "Phasellus",
                                "viverra",
                                "nulla",
                                "ut",
                                "metus",
                                "varius",
                                "laoreet",
                                "."
                            ],
                            [
                                "Quisque",
                                "rutrum",
                                "."
                            ],
                            [
                                "Aenean",
                                "imperdiet",
                                "."
                            ]
                        ],
                        "title": "2. Aenean leo"
                    }
                ],
                "text": [
                    []
                ],
                "title": "Terms and Conditions"
            }
        ],
        "extractionDate": [
            90309,
            8,
            41,
            11,
            28,
            7,
            2021
        ],
        "id": 1751660308386330068,
        "source": "file:///Users/tobias/Desktop/test.html",
        "title": "Terms and Conditions of Demo−Shop"
}
}
```

# C. Library Documentation

The methods provided by the developed library are presented in this chapter.

## C.1. Methods

`extractTandC()`

- Functionality: Extracts T&Cs of a single page and returns JSON as a string.
- Parameters:
  1. `url`: URL of the T&C page which shall be parsed
  2. `contentExtractor`: content extraction method (available options presented in Section C.2), *standard value:* *`NaiveStyleAndShortTextExclusion`*
  3. `threshold`: minimum coverage of MCS of lowest common ancestor required during content extraction, *standard value:* *`0.85`*
  4. `driver`: Selenium driver used to control a browser
- Return value: JSON as string

`extractTancD_multiple()`

- Functionality: Extract multiple T&Cs using the same driver and returns all JSON results as a list of strings.
- Parameters:
  1. `links`: list of URLs of the T&C pages which shall be parsed
  2. `contentExtractor`: content extraction method (available options presented in Section C.2), *standard value:* *`NaiveStyleAndShortTextExclusion`*
  3. `threshold`: minimum coverage of MCS of lowest common ancestor required during content extraction, *standard value:* *`0.85`*
  4. `driver`: Selenium driver used to control a browser
- Return value: List of JSON a strings

## C.2. Related

### Content Extractors

The different content extractors offer different methods of approximating the MCS. These methods were outlined in Section 4.4.

The following extractors are available:

- `NaiveStyle`: approximates the style by combining tag and attributes
  - `NaiveStyleAndShortTextExclusion`: requires a node to have at least 4 words (separated by white spaces) to be accounted for during MCS determination
- `RenderedStyle`: renders the actual CSS style *(performance intensive)*
  - `RenderedStyleAndShortTextExclusion`: requires a node to have at least 4 words (separated by white spaces) to be accounted for during MCS determination

### Selenium Driver

A Selenium driver for your browser version needs to be downloaded to your machine. The driver is then instantiated using:

```
from selenium import webdriver

driver = webdriver.<browser>(executable_path=<path>)
```

# List of Figures


1.1. AGB-Check Context . . . . . 2
1.2. RQ Relationship . . . . . 3

3.1. Withdrawal Form . . . . . 12
3.2. Hierarchy Styles (de) . . . . . 15
3.3. Hierarchy Styled (en) . . . . . 15
3.4. Table of Contents . . . . . 17

4.1. Architecture of Library . . . . . 19
4.2. Sequence Diagram of Library . . . . . 20
4.3. DOM-Tree incl. CSS . . . . . 24
4.4. Amount of Nodes (de) . . . . . 29
4.5. Amount of Nodes (en) . . . . . 30
4.6. Relation all Nodes vs. content Nodes (de) . . . . . 31
4.7. Hierarchy Tree . . . . . 32
4.8. Example Extraction . . . . . 33
4.9. Example Content Blocks . . . . . 34
4.10. Target Output Format . . . . . 39

B.1. Rendered Demo Shop (demo) . . . . . 53
B.2. Visual Separation, Step 1 (demo) . . . . . 60
B.3. Visual Separation, Step 2 (demo) . . . . . 61
B.4. Visual Separation, Step 3 (demo) . . . . . 61

# List of Tables


3.1. Extractor Performance (start) . . . . . 13
3.2. Extractor Performance (end) . . . . . 13
3.3. Extractor Performance (center) . . . . . 14

4.1. Coverage of Nodes in the German Sample . . . . . 27
4.2. Coverage of Nodes in the English Sample . . . . . 28

5.1. Evaluation Extractor Performance on Requirements Sample (start) . . . 41
5.2. Evaluation Extractor Performance on Requirements Sample (end) . . . . 42
5.3. Evaluation Extractor Performance on Test Sample . . . . . 42
5.4. Evaluation Hierarchy (Requirements Sample, de) . . . . . 45
5.5. Evaluation Hierarchy (Requirements Sample, en) . . . . . 45
5.6. Evaluation Hierarchy (Test Sample, de) . . . . . 47
5.7. Evaluation Hierarchy (Test Sample, en) . . . . . 47

A.1. German Sample . . . . . 51
A.2. English Sample . . . . . 52

B.1. MCS Results (demo) . . . . . 55
B.2. Lowest Common Ancestor Results (demo) . . . . . 56
B.3. BlockList (demo) . . . . . 59
B.4. MCS for Visual Separation (demo) . . . . . 60
B.5. Extracted Enumeration Patterns (demo) . . . . . 62

# Bibliography


[1] Y. Bakos, F. Marotta-Wurgler, and D. Trossen. "Does Anyone Read the Fine Print? Consumer Attention to Standard-Form Contracts." In: *The Journal of Legal Studies* 43 (Jan. 2014), pp. 1–35. DOI: 10.1086/674424.

[2] A. Barbaresi. "Generic Web Content Extraction with Open-Source Software." In: *Proceedings of the 15th Conference on Natural Language Processing, KONVENS 2019, Erlangen, Germany, October 9-11, 2019*. 2019.

[3] M. Baroni, F. Chantree, A. Kilgarriff, and S. Sharoff. "Cleaneval: A Competition for Cleaning Web Pages." In: Jan. 2008.

[4] D. Braun. "Automatic Semantic Analysis, Legal Assessment, and Summarization of Standard Form Contracts." PhD thesis. Technical University of Munich, July 2021.

[5] D. Braun and F. Matthes. "Automatic Detection of Terms and Conditions in German and English Online Shops." In: *16th International Conference on Web Information Systems and Technologies, WEBIST 2020*. SciTePress. 2020.

[6] D. Braun and P. Philip. *AI-Supported Legal Review of Terms and Conditions to Strengthen Consumer Protection*. 2020. URL: https://web.archive.org/web/20210506064242/https://wwwmatthes.in.tum.de/pages/1c5btiiss32n1/AI-Supported-Legal-Review-of-Terms-and-Conditions-to-Strengthen-Consumer-Protection-AGB-Check (visited on 05/06/2021).

[7] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. "VIPS: a Vision-based Page Segmentation Algorithm." In: (Jan. 2003).

[8] S. Debnath, P. Mitra, N. Pal, and C. Giles. "Automatic Identification of Informative Sections of Web Pages." In: *Knowledge and Data Engineering, IEEE Transactions on* 17 (Oct. 2005), pp. 1233–1246. DOI: 10.1109/TKDE.2005.138.

[9] S. Evert. "A Lightweight and Efficient Tool for Cleaning Web Pages." In: Jan. 2008.

[10] J. Gibson, B. Wellner, and S. Lubar. "Adaptive Web-Page Content Identification." In: Jan. 2007, pp. 105–112. DOI: 10.1145/1316902.1316920.

[11] F. V. Jiménez, M. A. Sánchez-Pérez, H. Gómez-Adorno, J. Posadas-Durán, G. Sidorov, and A. Gelbukh. "Improving the Boilerpipe Algorithm for Boilerplate Removal in News Articles Using HTML Tree Structure." In: *Computación y Sistemas* 22 (2018).

[12] A. Kilgarriff. "Last Words: Googleology is Bad Science." In: *Computational Linguistics* 33.1 (2007), pp. 147–151. DOI: 10.1162/coli.2007.33.1.147.

[13] C. Kohlschütter, P. Fankhauser, and W. Nejdl. "Boilerplate Detection Using Shallow Text Features." In: Oct. 2010, pp. 441–450. DOI: 10.1145/1718487.1718542.

[14] G. Lejeune and L. Zhu. "A New Proposal for Evaluating Web Page Cleaning Tools." In: *Computación y Sistemas* 22 (2018).

[15] M. Lui, T. Baldwin, and N. Vrl. "Cross-domain Feature Selection for Language Identification." In: (July 2021).

[16] T. Manabe and K. Tajima. "Extracting logical hierarchical structure of HTML documents based on headings." In: *Proceedings of the VLDB Endowment* 8 (Aug. 2015), pp. 1606–1617. DOI: 10.14778/2824032.2824058.

[17] J. Obar. "The Biggest Lie on the Internet: Ignoring the Privacy Policies and Terms of Service Policies of Social Networking Services." In: *SSRN Electronic Journal* (Jan. 2016). DOI: 10.2139/ssrn.2757465.

[18] H. Okada and H. Arakawa. "Automated extraction of non -tagged headers in webpages by decision trees." In: *SICE Annual Conference 2011*. 2011, pp. 2117–2120.

[19] J. Pasternack and D. Roth. "Extracting article text from the Web with maximum subsequence segmentation." In: Jan. 2009, pp. 971–980. DOI: 10.1145/1526709.1526840.

[20] J. Pomikálek. "Removing boilerplate and duplicate content from web corpora." PhD thesis. Masaryk University, Faculty of informatics, Brno, Czech Republic, 2011.

[21] H. Sano, S. Shiramatsu, T. Ozono, and T. Shintani. "A Web Page Segmentation Method based on Page Layouts and Title Blocks." In: (May 2021).